Explicit Preemption Placement for Real-Time Conditional Code

Bo Peng[1], Nathan Fisher[1], and Marko Bertogna[2]

[1]Department of Computer Science, Wayne State University, USA. Email: {fishern,et7889}@wayne.edu
[2]Algorithmic Research Group, University of Modena, Italy. Email: marko.bertogna@unimore.it

***Abstract*—In the limited-preemption scheduling model, tasks cooperate to offer suitable preemption points for reducing the overall preemption overhead. In the fixed preemption-point model, tasks are allowed to preempt only at statically defined preemption points, reducing the variability of the preemption delay and making the system more predictable. Different works have been proposed to determine the optimal selection of preemption points for minimizing the preemption overhead without affecting the system schedulability due to increased non-preemptivity. However, all works are based on very restrictive task models, without being able to deal with common coding structures like branches, conditional statements and loops.**

In this work, we overcome this limitation, by proposing a pseudo-polynomial-time algorithm that is capable of determining the optimal set of preemption points to minimize the worst-case execution time of jobs represented by control flowgraphs with arbitrarily-nested conditional structures, while preserving system schedulability. Exhaustive experiments are included to show that the proposed approach is able to significantly improve the bounds on the worst-case execution times of limited preemptive tasks.

***Index Terms*—limited-preemption scheduling; fixed preemption points; predictability; graph grammars; dynamic programming.**

I. Introduction

In worst-case execution time (WCET) analysis, an upper bound is calculated, for each job in the system, on the total aggregate amount of execution required to successfully complete the job. Real-time schedulability analysis has traditionally used the estimates derived from WCET analysis to determine whether every job in a system can be completed by its deadline. Thus, the effectiveness of the resulting schedulability analysis hinges upon the precision of WCET estimates. Unfortunately, many scheduler properties that simplify schedulability analysis often introduce pessimism into WCET analysis. For example, the oft-assumed property that jobs are arbitrarily preemptible leads to significant increase in the WCET estimates, as the analysis must assume that a preemption occurs often and the overhead of such preemption (due to context switch time and cache effects) must be added to the estimate.

To address this gap between WCET analysis and schedulability, recent work (e.g., Altmeyer et al. [3]) has focused on obtaining precise upper bounds on the cache-related preemption delays (CRPD) that a real-time job may experience due to cache evictions by preempting higher-priority tasks. This work has reduced the pessimism in the combined WCET and CRPD analysis, as it obtains a careful quantification of an upper bound on the number of cache block evictions a job could experience due to preemptions and also incorporates knowledge of how often a higher-priority job may preempt. However, another source of pessimism is the heterogeneous "cost" of preemption overhead within a job due to its cache-access patterns (i.e., spatial/temporal locality); Ramaprasad and Mueller [15] showed that CRPD costs varied depending upon on the location of the preemption point. Thus, a potential improvement in the combined WCET and CRPD analysis strategy might be obtainable by delaying the preemption of a job that is executing instructions with a high degree of spatial or temporal locality (in terms of memory access) until it reaches instructions with a lower level of memory locality.

Recently, Bertogna et al. [8] proposed such an approach that explicitly and efficiently determines (prior to runtime) the optimal choice of *explicit preemption points* (EPPs) in a job's code that minimize the preemption overhead while ensuring that system schedulability is not affected due to increased non-preemptivity. However, their proposed approach only deals with linear (non-branching) code and cannot handle jobs with control flow such as conditional statements (e.g., if-then-else statements) and loops. In this paper, we show how to extend the EPP approach of Bertogna et al. [8] for handling general conditional code with a solution that retains the efficient running time of the non-branching version of the problem. We believe that such extensions are absolutely necessary for the EPP approach to be widely applicable and useful to a real-time system designer.

II. Model

We refer to the problem of determining the optimal choice of EPPs for a program (i.e., job) as the *explicit preemption placement* problem. To model the explicit preemption placement problem, we assume that a program $\mathcal{P}$ has been divided into a set of non-preemptive basic blocks (BBs). A basic block represents a subset of sequential code for which a WCET estimate has been obtained. (Note that the work in [8] assumes that any conditional code is entirely contained within a single BB. Also note that we depart slightly from the standard

This research has been supported in part by an NSF CAREER Grant (CNS-0953585), an NSF CSR Grant (CNS-1116787), and the European Commission under the P-SOCRATES project (FP7-ICT-611016).

definition of BB which specifies that a BB is a maximal sequence of sequential code; for generality, our definition of BB permits the designer to break a single traditional BB into smaller segments.) Let $V = \{\delta_1, \delta_2, \ldots, \delta_n\}$ be the set of basic blocks. A *control flowgraph* $G_{\mathcal{P}} = (V, E, \delta_s, \delta_z)$ describes the control flow of $\mathcal{P}$. A flowgraph is a directed graph (V, E) with a distinguished start vertex (basic block) δ_s and exit vertex (basic block) δ_z such that for $\delta_v \in V$ there is a path from δ_s to δ_v and a path from δ_v to δ_z. Clearly, each vertex $\delta_v \in V$ represents a BB in $\mathcal{P}$. An edge $(\delta_u, \delta_v) \in E \subseteq V \times V$ means that the execution of BB δ_u immediately precedes the execution of BB δ_v in some execution path of $\mathcal{P}$, and that a preemption is permitted between the two BBs, i.e., E is the set of possible EPPs. A path p is an ordered set of consecutive vertices, such that each vertex in p has an edge from its predecessor. Let $\mathsf{paths}(G_{\mathcal{P}}, \delta_x, \delta_y)$ be the set of possible execution paths between the basic blocks δ_x and δ_y in the control flowgraph $G_{\mathcal{P}}$. We say that $\delta_u \preceq_p \delta_v$, if δ_u precedes (or equals) δ_v along path $p \in \mathsf{paths}(G_{\mathcal{P}}, \delta_x, \delta_y)$. The operator $\prec_p$ denotes strict precedence.

For this paper, we will study an important class of control flowgraphs called *series-parallel graphs* [18]. That is, $G_{\mathcal{P}}$ can be obtained by applying some sequence of the following three operations:

1) Graph Creation: create a graph with two nodes and a single directed edge between them.
2) Series Composition: given two series-parallel graphs $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$, a new graph is created by merging the sink node of $G_{\mathcal{X}}$ with the source node of $G_{\mathcal{Y}}$.
3) Parallel Composition: given two series-parallel graphs $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$, a new graph is created by merging (i) the source nodes of $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$, and (ii) the sink nodes of $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$.

Section V-B will give an equivalent grammar-based specification for construction of a series-parallel graph $G_{\mathcal{P}}$. Note that previous work on explicit preemption placement [8] handled only graphs created using the series composition operation.

Series-parallel graphs are appropriate to model many structured programming language constructs such as `if-then-else` statements, `switch` statements, and loops with bounded iterations [2] where programs can easily be subdivided into segments with a single entry point and a single exit point. As acknowledged in [17], "real-time systems only use a restricted form of programming, which guarantees that programs always terminate, recursion is explicitly bounded or not allowed, as are the iteration counts of loops". Therefore, the adopted task model based on series-parallel graphs can be efficiently used to model real-time tasks implemented via a structured programming language. Note that loops may be modeled as series compositions of multiple blocks. A long loop can then be simply split into the series composition of different sub-loops, each one mapped to a basic block of smaller granularity. The size of such sub-loops can be freely decided at design time. The appendix of this paper also addresses non-unrolled loops and non-inline functions directly without requiring mapping these structures into a larger basic block.

The only limitation of the series-parallel graph model is in the preclusion of `goto` statements and early returns. However, as proved in [9], every structured program can be expressed as a combination of sequential instructions, conditional branches and loops, without needing `goto`'s and multiple exit points.

We assume the WCET of a BB is given by a function $C : V \mapsto \mathbb{R}_{\geq 0}$, and that a function $\xi : E \mapsto \mathbb{R}_{\geq 0}$ is given, providing an upper bound of the preemption overhead at each potential EPPs. This latter function incorporates the contributions due to CRPD, pipeline flushing and scheduling cost of the target system. When the number of BBs is not too large, both functions can be computed in reasonable time exploiting the features of modern timing analysis tools on a given task *executed in isolation*.

If preemption is not permitted between basic blocks $\delta_u, \delta_v \in V$ for edge $(\delta_u, \delta_v) \in E$, we can model this scenario by setting $\xi(\delta_u, \delta_v)$ equal to ∞. We assume that there are two "dummy" *sentinel basic blocks* $\delta_{-\infty}$ and δ_{∞} such that $(\delta_{-\infty}, \delta_s)$ and $(\delta_z, \delta_{\infty})$ are in E and $C(\delta_{-\infty}) = C(\delta_{\infty}) = \xi(\delta_{-\infty}, \delta_s) = \xi(\delta_z, \delta_{\infty}) = 0$. Edges $(\delta_{-\infty}, \delta_s)$ and $(\delta_z, \delta_{\infty})$ are called *sentinel edges*.

III. Problem Statement

Let $\mathcal{G}$ be the set of flowgraphs that define programs with conditional control structures according to the above model. Since the selection of EPPs will create non-preemptible regions in $\mathcal{P}$, the schedulability of the system is affected by a choice of EPPs. In previous work by Baruah [4] and Bertogna [5], schedulability techniques were derived to obtain a constant Q which quantifies the maximum duration of any non-preemptive region in $\mathcal{P}$; we use this constant Q in our model.

Given the above model, our goal is to find a selection of EPPs that minimizes the WCET of $\mathcal{P}$, without imposing non-preemptive regions that are greater than the maximum allowed non-preemptive execution Q. More formally, the main problem addressed in this paper is the following.

Problem statement:
Given $G_{\mathcal{P}} \in \mathcal{G}$ and associated functions ξ and C, find the optimal set $S \subseteq E$ that minimizes

$$\Phi(G_{\mathcal{P}}, S) \stackrel{def}{=} \max_{p \in \mathsf{paths}(G_{\mathcal{P}}, \delta_s, \delta_z)} \left\{ \sum_{\delta_u \in p} C(\delta_u) + \sum_{\substack{\delta_u, \delta_v \in p \\ (\delta_u, \delta_v) \in S}} \xi(\delta_u, \delta_v) \right\} \quad (1)$$

subject to the constraint that $\forall p \in \mathsf{paths}(G_{\mathcal{P}}, \delta_s, \delta_z), \delta_i \in p$: $\exists e_1 = (\delta_u, \delta_v), e_2 = (\delta_x, \delta_y) \in S$ ::

$$(\delta_u \preceq_p \delta_i \preceq_p \delta_y) \quad \wedge \quad \left(\xi(e_1) + \sum_{\substack{\delta_j \in p \\ \delta_v \preceq_p \delta_j \preceq_p \delta_x}} C(\delta_j) \leq Q \right). \quad (2)$$

In words, the last constraint means that for each path p in paths and for any basic block $\delta_i \in p$, there exist two EPPs

in S such that the non-preemptive region between such EPPs contains δ_i and has a total execution cost no larger than the schedulability constraint Q. If no such S exists, we say that $G_{\mathcal{P}}$ is not feasible for the given Q. In the above problem statement, the Φ function corresponds to the WCET of $\mathcal{P}$ when we have the non-preemptivity constraint of Q.

One note about solutions for the above problem: we seek *optimal* solutions under the assumption that a preemption will occur at each EPP (i.e., the worst case will occur). Clearly, if the system scheduler is lightly loaded and can tolerate non-preemptively executing a task for more than Q time units, some other selection of preemption points may result in lower overhead at runtime. However, since we are interested in knowing at design-time the WCET and minimizing the contribution of preemptions to the WCET, this notion of optimality is the most natural and appropriate.

IV. Related Work

The research on limited preemptive scheduling algorithms has recently received significant attention due to the performance benefits in terms of reduced preemption overhead and increased predictability. According to this scheduling model, each task is divided into a set of non-preemptive regions, so that preemptions can take place only at a subset of points. Depending on the location of such non-preemptive regions, limited preemptive schedulers are divided into two sub-categories: fixed and floating preemption point models.

The fixed preemption point model has been introduced by Burns [11]. According to this model, tasks are divided into statically defined non-preemptive chunks of fixed length, so that preemptions are allowed only at chunks' boundaries. Since tasks cooperate in order to offer suitable preemption points to decrease the context switch overhead, such a model is also called Cooperative Scheduling. A tight schedulability analysis for the fixed preemption point model has been presented by Bril et al. [10].

In the floating preemption point model, instead, the size and location of the non-preemptive regions are not known before run-time, but are determined during task execution. Only an upper bound is given on the maximum allowed non-preemptive region Q of a task. The floating model has been adopted by Baruah et al. [4], [5] for EDF scheduled systems, and by Yao et al. [19] for Fixed Priority scheduled systems. In both works, upper bounds are provided on the largest non-preemptive region Q that can be enforced in each task without causing any deadline miss.

The problem of computing the optimal Q in the fixed preemption point model has been addressed by Bertogna et al. [7] for fixed priority systems, showing that inserting a non-preemptive region at the end of a task might increase the schedulability of the system with respect to fully preemptive or non-preemptive scheduling. In the same paper, an algorithm is shown to compute the largest non-preemptive region Q for each task in order to maximize the schedulability of the system. Davis et al. [13] later adapted this method, showing an optimal priority assignment to be used in combination with the fixed preemption point model.

When preemption overhead is included in the analysis, the derived bounds on the largest non-preemptive region Q of each task can be exploited to reduce as much as possible the preemption overhead without compromising feasibility. The fixed preemption point model can be adopted to insert preemption points at suitable locations, such that the length of each non-preemptive region does not exceed Q. In [6], an optimal preemption point placement method with linear complexity is presented, under the assumption that the preemption overhead is constant throughout the code of each task. In [8], this assumption has been relaxed, considering a variable preemption overhead and selecting the optimal subset of preemption points that maximizes the schedulability of the task system. The preemption point placement algorithm has pseudo-polynomial complexity, proportional to Q and to the number of potential preemption points. In both papers, a linear task structure is considered, so that each task is composed of a linear sequence of basic blocks, and if-then-else constructs are entirely contained inside a BB. An exhaustive survey on the existing limited preemptive scheduling techniques is available in [12].

V. Flowgraphs for Real-Time Conditional Code

The definition of control flowgraph is broad enough to permit a wide variety of programmatic structures. In this paper, we are focused on analyzing programs that have conditional control structures such as `if-then-else` or `switch` constructs. To be precise, we need to be specific about the types of control flowgraphs that we will address in this paper. Typically, a programming language designer defines a *context-free grammar* to specify the set of strings that are considered valid programs in the programming language. However, in this paper, our input is a control flowgraph $G_{\mathcal{P}}$; thus, we need to specify the subset of control flowgraphs for which our approach applies. For this purpose, we create a *flowgraph grammar* [14] over control flowgraphs to specify the set of graphs (i.e., programs) that we consider valid conditional real-time control flowgraphs. In this section, we first give a brief background on graph grammars. Then, we give the formal specification of the graph grammar for the real-time conditional code considered in this paper.

A. Background: Graph Grammars

The concept of *graph grammars* was introduced as an attempt to formalize the identification of different programmatic structures and develop analysis tools for the purpose of code optimization [14]. In this paper, we utilize the concept of graph grammars for two reasons: 1) graph grammars provide a formalism for automatically describing and recognizing the set of control flowgraphs that satisfy our task model; and 2) they permit an elegant means of expressing an algorithm to solve the EPP problem for real-time conditional code.

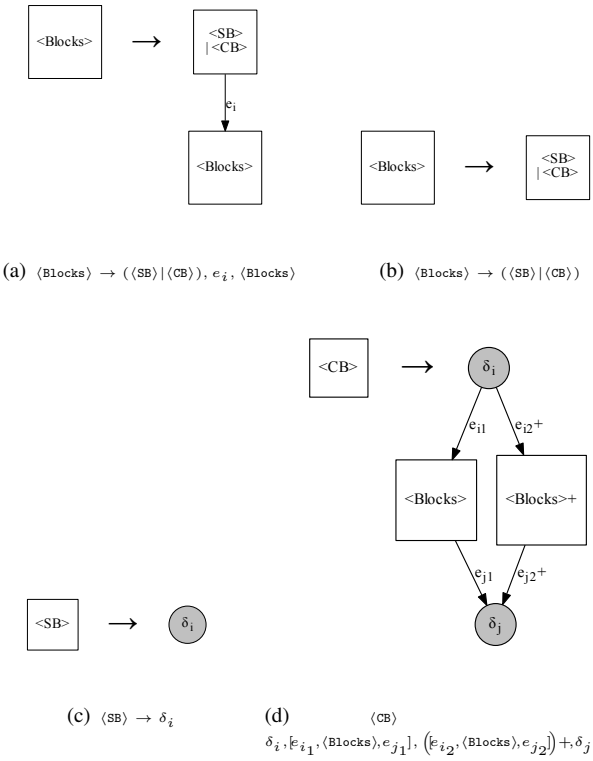
A graph grammar is the means of specifying the "syntax" of proper control flowgraphs. Like a textual program, a control

flowgraph contains tokens; however, instead tokens being strings of number, letters, or symbols, as in a textual program, the set of tokens in a control flowgraphs are vertices and edges. The graph grammar is the set of rules specifying how these "tokens" may be combined to form a valid control flowgraph. Each rule in a graph grammar is called a *production*. A production rule has a left-hand side and a right-hand side. The left-hand side of a production contains a *non-terminal node*; the non-terminal node is an abstract representation of some collection of vertices and edges in the control flowgraph. The right-hand side of the production contains non-terminal node(s) and/or *terminal node(s)*. Each terminal node is a vertex in the control flowgraph and, in our setting, it represents a BB. The production specifies that the non-terminal node on the left-hand side of the production may be substituted (or rewritten) with the nodes on the right-hand side of the production; this process of substituting is called a *derivation*. A graph grammar $\mathcal{G}$ is a set of production rules which define a specific graph language $L(\mathcal{G})$ generated by graph grammar $\mathcal{G}$. A graph G is in the language $L(\mathcal{G})$, if there exists a sequence of derivations, starting from a specified starting non-terminal node, that uses productions of $\mathcal{G}$ and results in graph G. A graph grammar is called *context-free* if each production has only non-terminal nodes on the left-hand side of the production. A graph grammar is called *unambiguous* if for each $G \in L(\mathcal{G})$ there exists a unique sequence of derivations for G. In the next subsection, we give the production rules for our control flowgraph grammar of real-time conditional code.

B. Real-time Conditional Flowgraph Grammar Specification

Figure 1 gives the production rules for graph grammar $\mathcal{G}$ for control flowgraphs that satisfy our task model. A boxed node represents a non-terminal node; a circle node represents a basic block (i.e., a terminal node). We can also represent the graph grammar by a traditional text-based grammar. The text-based equivalent production is given along with the graph grammar production. We use *Extended Backus-Naur Form* (EBNF) to describe both the graph grammar $\mathcal{G}$ and its equivalent text-based representation. A term in angle brackets (e.g., $\langle$`Blocks`$\rangle$) indicates that this is a non-terminal node. The $x \mid y$ operator means that either the term x or y may be used in the production. The expression $x+$ indicates that one or more successive occurrences of the expression x may be used in the substitution. Any programming languages textbook will contain more complete and formal descriptions of EBNF and the terminology of Section V-A (as it applies to textual grammars). For a good textbook on programming languages and their grammars, we refer the reader to Scott [16].

Figure 1(a) shows that a set of blocks (represented by $\langle$`Blocks`$\rangle$) consists of either a single block ($\langle$`SB`$\rangle$) or a conditional block ($\langle$`CB`$\rangle$), connected by edge e_i to another set of blocks. Figure 1(b) indicates that a set of blocks could just comprise a single sequential or conditional block without any subsequent blocks. Figure 1(c) shows that a single block is a simple BB δ_i. Finally, Figure 1(d) gives the production for conditional blocks: a conditional block has a forking BB



(a) $\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle \mid \langle \texttt{CB} \rangle), e_i, \langle \texttt{Blocks} \rangle$ (b) $\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle \mid \langle \texttt{CB} \rangle)$

(c) $\langle \texttt{SB} \rangle \rightarrow \delta_i$ (d) $\langle \texttt{CB} \rangle \rightarrow \delta_i, \left(e_{i_1}, \langle \texttt{Blocks} \rangle, e_{j_1}\right], \left(e_{i_2}, \langle \texttt{Blocks} \rangle, e_{j_2}\right]+, \delta_j$

Fig. 1. Production rules for control flowgraph grammar $\mathcal{G}$.

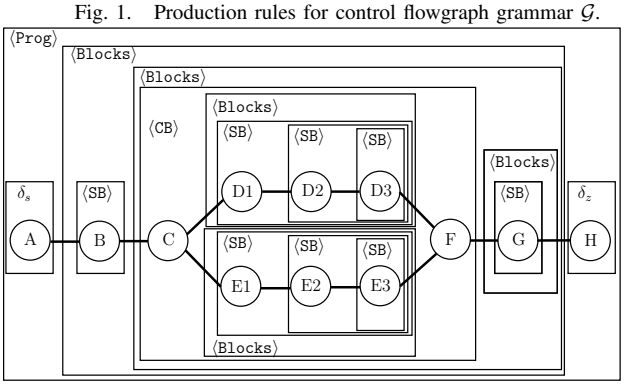


Fig. 2. A derivation using the production rules of Figure 1 over an example control flowgraph.

δ_i with two or more outgoing edges and a joining BB δ_j with an equivalent number of incoming edges. If there are k paths from δ_i to δ_j in the conditional block, then for any path p_ℓ ($1 \leq \ell \leq k$) a collection of blocks is contained on the path between δ_i and δ_j and connected to these BBs by edges e_{i_ℓ} and e_{j_ℓ}, respectively. Clearly, this grammar permits multiple nested levels of conditional blocks, as the blocks within each path may consist of additional conditional blocks. Note that a conditional block with two paths can represent an `if-then-else` construct in a program; a conditional block with two or more paths can represent a `switch` construct. It can easily be seen that $\mathcal{G}$ represents an unambiguous context-free grammar. Figure 2 shows an application of the production rules given in Figure 1.

VI. Dynamic-Programming Algorithm

As mentioned in Section IV, an optimal preemption point placement method has been presented in [8] for linear task structures, i.e., without any conditional block. One might wonder whether an optimal solution can be obtained by applying the method in [8] to the worst-case path of a conditional task structure. Or, alternatively, by repeatedly applying it to each possible path in a conditional task structure. Unfortunately, this is not the case, since the "globally" optimal solution (i.e., the EPP placement resulting in the smallest possible WCET) might differ from the combination of the optimal solutions of each path. Consider the example in Figure 3, where a simple task structure including a conditional branch is depicted. The WCET of each BB and the preemption cost at each edge are specified above each node and each edge, respectively. The maximum allowed non-preemptive section is assumed to be $Q = 8$.

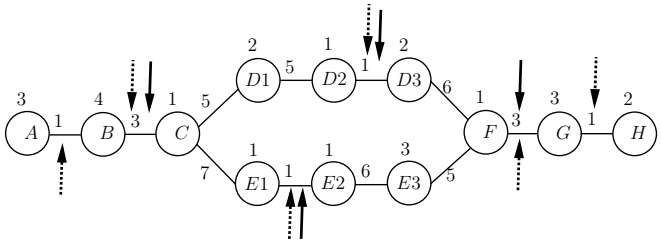


Fig. 3. Example of EPP selection using the linear method in [8] for each path (dashed arrows) w.r.t. the optimal EPP selection (solid arrows).

If the linear method in [8] is applied to the upper branch, three EPPs are placed: one between nodes B and C, one between $D2$ and $D3$, and the last one between G and H. When the same method is applied to the lower branch, one EPP is placed between nodes A and B, one between $E1$ and $E2$, and another one between F and G. Altogether, six different EPPs are placed in the graph, leading to an overall preemption overhead of 9 time-units in each of the paths, and a total WCET of 28 time-units. However, a smarter placement of EPPs is able to reach a smaller WCET by taking a "global" view of the task structure. A WCET of 26 can be obtained selecting four EPPs: one between nodes B and C, one between $D2$ and $D3$, one between $E1$ and $E2$, and the last one between F and G, with a total preemption cost of 8 time-units in each of the paths. Note that the selected placement does not coincide with any of the two linear selections.

In this section, we describe a dynamic-programming-based approach for obtaining an optimal solution to the EPP placement problem. Our returned solution $S^* \subseteq E$ is optimal in the sense that any other $S' \subseteq E$ must result in a higher or equal WCET for $\mathcal{P}$; in other words, $\Phi(G_{\mathcal{P}}, S^*) \leq \Phi(G_{\mathcal{P}}, S')$ where Φ is the objective function defined in Equation (1) of our problem statement. In the previous section, each non-terminal node (i.e., ⟨SB⟩, ⟨CB⟩, and ⟨Blocks⟩) consists of a collection of basic blocks and other non-terminal nodes. Thus, there is a well-defined structure that we may potentially exploit to determine the optimal placement of EPPs within program $\mathcal{P}$.

We first give a high-level overview of our approach for solving the EPP problem in the following subsection. Afterwards, we give a recursive formulation of the EPP placement problem by exploiting the structure defined by the graph grammar for conditional real-time control flowgraphs. Finally, we describe the bottom-up computation of the optimal placement of EPPs. The appendix contains extensions of our dynamic-programming approach for dealing with non-unrolled loops and non-inline functions.

A. High-Level Overview of Approach

The basic idea behind our approach is that we can compute the optimal EPPs for subgraphs in the program's flowgraph and utilize the subgraph solutions to determine the overall optimal EPP solution for the entire flowgraph. At a high level, we will begin by determining the EPP for innermost conditional blocks first and then work our way to the outermost flowgraph structures. For instance, in the example flowgraph in Figures 2 and 3, the optimal choice of EPPs for the conditional block beginning at basic block C and ending at basic block F will be determined prior to calculating the EPPs for the overall flowgraph. Unfortunately, it is not sufficient to calculate a single EPP solution for a substructure due to the dependence on EPP placement with EPP selected by the larger structure; for instance, the optimal choice of EPPs for the conditional block in Figure 2 depends upon the last preemption that occurs before basic block C (e.g., is (A, B) or (B, C) the last preemption before C?) and the first preemption selected after basic block F (e.g., is (F, G) or (G, H) the first preemption after F?). The reason for this dependence is that the choice of preemptions within the nested structure must satisfy the constraint in the larger structure that EPPs are no further than Q units apart (i.e., the constraint of Equation 2).

A natural question at this point is: *how do we compute the optimal EPPs for inner substructures first when the preemption placement is dependent upon the EPP selection in outermost structures?* The answer is to compute and store the optimal EPP selection/cost for all possible preemption values before and after each substructure. In other words, assuming for any substructure $G^A_{\mathcal{P}}$ of the input flowgraph that (i) the last preemption before substructure $G^A_{\mathcal{P}}$ occurs ζ_1 time before the first basic block of $G^A_{\mathcal{P}}$, and (ii) the earliest preemption after $G^A_{\mathcal{P}}$ occurs ζ_2 time units after the last basic block of $G^A_{\mathcal{P}}$. Then, a set $S^A(\zeta_1, \zeta_2)$ and cost-matrix $\mathsf{cost}(G^A_{\mathcal{P}}, \zeta_1, \zeta_2)$ will be computed for all possible values of ζ_1 and ζ_2. The set $S^A(\zeta_1, \zeta_2)$ represents the optimal EPP selection (w.r.t. Equations 1 and 2) for substructure $G^A_{\mathcal{P}}$ given the preemptions' times before and after $G^A_{\mathcal{P}}$ are ζ_1 and ζ_2. The cost-matrix $\mathsf{cost}(G^A_{\mathcal{P}}, \zeta_1, \zeta_2)$ represents the corresponding WCET+CRPD cost for substructure $G^A_{\mathcal{P}}$ with EPP selection $S^A(\zeta_1, \zeta_2)$. For instance, in the example of Figure 3, when $G^A_{\mathcal{P}}$ represents the conditional block staring at C and ending at F, the set $S^A(3, 3)$ equals $\{(D2, D3), (E1, E2)\}$ and $\mathsf{cost}(G^A_{\mathcal{P}}, 3, 3)$ equals 8.

Given the above approach, the reader may also wonder: *how many subproblems for each substructure need to be solved?* By the constraint of Equation 2, the largest value

for preemptions before or after any substructure (i.e., upper bounds for values of ζ_1 and ζ_2) is Q time units. Furthermore, if we assume that due to the clock tick granularity of the system preemption times must be integers, then there are at most $Q \times Q = Q^2$ combinations of ζ_1 and ζ_2; thus, there are at most Q^2 subproblems we must compute for each substructure. In the next subsection, we obtain a recursive formulation which computes the optimal EPP selection for any substructure based on the optimal solutions to the subsubproblems. This is a classic dynamic programming approach. After obtaining the recursive formulation, Subsection VI-C will describe a more efficient bottom-up implementation.

B. Optimal Substructure & Recursive Formulation

A derivation of $\mathcal{G}$ on an input graph $G_{\mathcal{P}}$ is a sequence of production rules applied to the input graph. During the derivation, the underlying basic blocks are assigned to the non-terminal nodes (see Figure 2 for an example). Let A be any production in the derivation for $G_{\mathcal{P}}$. Let $G^A_{\mathcal{P}}$ denote the subgraph induced by production A in $G_{\mathcal{P}}$ derivation over grammar $\mathcal{G}$. In other words, $G^A_{\mathcal{P}} = (V^A, E^A)$ is the collection of basic blocks and edges represented by the non-terminal block on the left-hand side of production A.

For series-parallel graphs, it may be shown [18] that for any production A, there exists for $G^A_{\mathcal{P}}$ exactly one node (i.e., basic block) with in-degree equal to zero and exactly one node with out-degree equal to zero. Let $\mathsf{InBlock}(G^A_{\mathcal{P}})$ denote the node of $G^A_{\mathcal{P}}$ with in-degree equal to zero and $\mathsf{OutBlock}(G^A_{\mathcal{P}})$ as the node with out-degree equal to zero.

We now define a modified version of $G^A_{\mathcal{P}}$ called $G^A_{\mathcal{P}}(\zeta_1, \zeta_2) = (V^A(\zeta_1, \zeta_2), E^A(\zeta_1, \zeta_2))$ where

$$V^A(\zeta_1, \zeta_2) \stackrel{\text{def}}{=} V^A \cup \{\delta^A_s, \delta^A_z\} \tag{3}$$

and

$$E^A(\zeta_1, \zeta_2) \stackrel{\text{def}}{=} E^A \cup \left\{ \begin{array}{l} e^A_s \stackrel{\text{def}}{=} (\delta^A_s, \mathsf{InBlock}(G^A_{\mathcal{P}})), \\ e^A_z \stackrel{\text{def}}{=} (\mathsf{OutBlock}(G^A_{\mathcal{P}}), \delta^A_z) \end{array} \right\}. \tag{4}$$

Furthermore, $C(\delta^A_s) = \zeta_1$, $C(\delta^A_z) = \zeta_2$, and $\xi(e^A_s) = \xi(e^A_z) = \infty$. Intuitively, $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ is an augmentation of $G^A_{\mathcal{P}}$ by adding non-preemptible BBs δ^A_s and δ^A_z with size ζ_1 and ζ_2 to the beginning and end (respectively) of $G^A_{\mathcal{P}}$. Let $S^A(\zeta_1, \zeta_2) \subseteq E^A(\zeta_1, \zeta_2)$ be the optimal set of EPPs for $G^A_{\mathcal{P}}$ according to the objective function of Equation (1) respecting the constraints of Equation (2). We will next show that we can obtain an optimal solution to the EPP placement problem for $G_{\mathcal{P}}$ by first optimally solving the EPP placement problem for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ for all productions A in a derivation for $G_{\mathcal{P}}$ and for all values of $\zeta_i \in \{0, 1, \ldots, Q-1\}$ (where $i = 1, 2$).

We first consider production (a):

$$\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle \mid \langle \texttt{CB} \rangle), e_i, \langle \texttt{Blocks} \rangle.$$

In a general application of production (a), we let $G^A_{\mathcal{P}}$ denote the non-terminal block on the left-hand side of the rule, and $G^B_{\mathcal{P}}$ and $G^C_{\mathcal{P}}$ denote the first and second non-terminal blocks, respectively, on the right-hand side of the rule:

$$G^A_{\mathcal{P}} \rightarrow G^B_{\mathcal{P}}, e_i, G^C_{\mathcal{P}},$$

where $e_i = (\mathsf{OutBlock}(G^B_{\mathcal{P}}), \mathsf{InBlock}(G^C_{\mathcal{P}}))$.

Theorem 1. *When applying production (a) over feasible $G_{\mathcal{P}}$ and Q, an optimal set $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$) is equal to*

$$S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2), \tag{5}$$

or one of the following for $x \in \{1, \ldots, Q-1\}$:

$$S^B(\zeta_1, x) \cup S^C(Q - x, \zeta_2). \tag{6}$$

Proof: The proof is by contradiction. There are two cases dependent upon whether $e_i \in S^A(\zeta_1, \zeta_2)$ is true. Let us first consider the case where e_i is in the set $S^A(\zeta_1, \zeta_2)$.

Case 1: $[e_i \in S^A(\zeta_1, \zeta_2)]$

Assume there exist $S' \subseteq E^B(\zeta_1, 0)$ and $S'' \subseteq E^C(\xi(e_i), \zeta_2)$, such that $S^A(\zeta_1, \zeta_2)$ equals $S' \cup \{e_i\} \cup S''$, and $\Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S' \cup \{e_i\} \cup S''\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2)\right)$. In words, $S^B(\zeta_1, 0)$ and $S^C(\xi(e_i), \zeta_2)$ are optimal sets of EPPs for graphs $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively, but $S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2)$ is not an optimal set of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$.

We will show that the following properties are satisfied:

P1: S' and S'' satisfy the constraints of Equation (2) for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively.

P2: At least one of the following strict inequalities holds:
$\Phi(G^B_{\mathcal{P}}(\zeta_1, 0), S') < \Phi(G^B_{\mathcal{P}}(\zeta_1, 0), S^B(\zeta_1, 0))$, or
$\Phi(G^C_{\mathcal{P}}(\xi(e_i), \zeta_2), S'') < \Phi(G^C_{\mathcal{P}}(\xi(e_i), \zeta_2), S^C(\xi(e_i), \zeta_2))$.

Property 1 implies that S' and S'' are feasible solutions to the EPP placement problem for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively. Property 2 implies that at least one between $S^B(\zeta_1, 0)$ and $S^C(\xi(e_i), \zeta_2)$ cannot be an optimal solution for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively, reaching a contradiction.

Proof of P1: To prove Property 1, note that $S' \equiv S^A(\zeta_1, \zeta_2) \cap E^B(\zeta_1, 0)$. Since $S^A(\zeta_1, \zeta_2)$ satisfies the EPP constraint for the graph $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$, and the portion of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ before e_i coincides with $G^B_{\mathcal{P}}(\zeta_1, 0)$ (an exit node of length zero can be simply ignored), then also S' satisfies the EPP constraint of Equation (2) for $G^B_{\mathcal{P}}(\zeta_1, 0)$.

Some more steps are needed to prove the same property for S'' and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$. Note that $S'' \equiv S^A(\zeta_1, \zeta_2) \cap E^C(\xi(e_i), \zeta_2)$. For each path p entering $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, let $e^p_{\text{first}} = (\delta^p_x, \delta^p_y)$ be the first EPP of S'' in p. Since $S^A(\zeta_1, \zeta_2)$ satisfies the EPP constraint for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$, and the portion of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ after δ^p_x coincides with that of $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, all basic blocks of $V^C(\xi(e_i), \zeta_2)$ that come after δ^p_x in path p continue to satisfy the EPP constraint of Equation (2) for S'' in $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$. Thus, we just need to prove that the remaining basic blocks ($\delta_j \in V^C(\xi(e_i), \zeta_2) : \delta_j \preceq_p \delta^p_x$) also satisfy the EPP constraint for S'' in $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$:

$$\xi((\delta_{-\infty}, \delta^C_s)) + \sum_{\substack{\delta_j \in p \\ \delta^C_s \preceq_p \delta_j \preceq_p \delta^p_x}} C(\delta_j) \leq Q. \tag{7}$$

By convention for the sentinel edges, $\xi((\delta_{-\infty}, \delta^C_s))$ equals zero. Since $S'' \cup \{e_i\}$ satisfies the EPP constraint for

$G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$, it must be that

$$\xi(e_i) + \sum_{\substack{\delta_j \in p \\ \mathsf{InBlock}(G^C_{\mathcal{P}}) \preceq_p \delta_j \preceq_p \delta^p_x}} C(\delta_j) \leq Q. \tag{8}$$

However, observe that $C(\delta^C_s)$ equals $\xi(e_i)$ by definition of $G^C_{\mathcal{P}}(\xi(e_i),\zeta_2)$, and the remaining blocks are $\delta_j \in p : \mathsf{InBlock}(G^C_{\mathcal{P}}) \preceq_p \delta_j \preceq_p \delta_x$. Thus, Equation (8) implies that Equation (7) is true. Hence, Property 1 holds also for S'' and $G^C_{\mathcal{P}}(\xi(e_i),\zeta_2)$. ∎

Proof of P2: To prove Property 2, remember that we assumed: $\Phi\left(G^A_{\mathcal{P}}(\zeta_1,\zeta_2), S' \cup \{e_i\} \cup S''\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1,\zeta_2), S^B(\zeta_1,0) \cup \{e_i\} \cup S^C(\xi(e_i),\zeta_2)\right)$.

The considered graph might have multiple paths, all of which pass through e_i. Since $G^B_{\mathcal{P}}$ and $G^C_{\mathcal{P}}$ are disjoint sets, the smaller Φ function obtained using $S' \cup \{e_i\} \cup S''$ instead of $S^B(\zeta_1,0) \cup \{e_i\} \cup S^C(\xi(e_i),\zeta_2)$ is due to a better EPP selection in S' than in $S^B(\zeta_1,0)$, or in S'' than in $S^C(\xi(e_i),\zeta_2)$. In the first case, since the portion of $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ before e_i coincides with $G^B_{\mathcal{P}}(\zeta_1,0)$ (ignoring the exit node of length zero), it follows that $\Phi(G^B_{\mathcal{P}}(\zeta_1,0),S') < \Phi(G^B_{\mathcal{P}}(\zeta_1,0),S^B(\zeta_1,0))$, proving the first inequality of Property 2. In the second case, since the portion of $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ after e_i coincides with the portion of $G^C_{\mathcal{P}}(\xi(e_i),\zeta_2)$ after δ^C_s, and $C(\delta^C_s) = \xi(e_i)$, it must be that $\Phi(G^C_{\mathcal{P}}(\xi(e_i),\zeta_2),S'') < \Phi(G^B_{\mathcal{P}}(\xi(e_i),\zeta_2),S^B(\xi(e_i),\zeta_2))$, proving the second inequality of Property 2. ∎

The case where e_i is in the set $S^A(\zeta_1,\zeta_2)$ stands proved. It remains to prove the other case. ∎

Case 2: $[e_i \notin S^A(\zeta_1,\zeta_2)]$

P1: S' and S'' satisfy the constraints of Equation (2) for $G^B_{\mathcal{P}}(\zeta_1,x)$ and $G^C_{\mathcal{P}}(Q-x,\zeta_2)$, respectively.

P2: At least one of the following strict inequalities holds: $\Phi(G^B_{\mathcal{P}}(\zeta_1,x),S') < \Phi(G^B_{\mathcal{P}}(\zeta_1,x),S^B(\zeta_1,x))$, or $\Phi(G^C_{\mathcal{P}}(Q-x,\zeta_2),S'') < \Phi(G^C_{\mathcal{P}}(Q-x,\zeta_2),S^C(Q-x,\zeta_2))$.

Since, by P1, S' and S'' are feasible solutions to the EPP placement problem for $G^B_{\mathcal{P}}(\zeta_1,0)$ and $G^C_{\mathcal{P}}(\xi(e_i),\zeta_2)$, respectively, and, by P2, if either $S^B(\zeta_1,0)$ or $S^C(\xi(e_i),\zeta_2)$ cannot be an optimal solution for $G^B_{\mathcal{P}}(\zeta_1,0)$ and $G^C_{\mathcal{P}}(\xi(e_i),\zeta_2)$, respectively, then a contradiction is reached (similar to Case 1), proving also Case 2. Note that the range of x from 1 to $Q-1$ is to maintain the NPR between $G^B_{\mathcal{P}}(\zeta_1,x)$ and $G^C_{\mathcal{P}}(Q-x,\zeta_2)$. 0 and 1 should not be considered since x with value 0 is exactly the first case, and given x equal to Q would cause the CRPD+WCET of the last NPR in $G^B_{\mathcal{P}}(\zeta_1,x)$ larger than Q. ∎

Having proved that in both considered cases one among Equations (5) and (6) holds, the Theorem stands proved. ∎

We have just shown that optimal substructure exists for determining the optimal EPP placement for any subgraph that corresponds to a single block production in a derivation of $G_{\mathcal{P}}$. We can exploit this optimal substructure to obtain a recursive formulation for quantifying the optimal value of Φ for the subgraph. We first define some notation.

Let $\mu_a(\text{expr})$ return a if expr is true and one if false. Moreover, let the function $\mathsf{cost}(A,\zeta_1,\zeta_2)$ be

$$\Phi(G^A_{\mathcal{P}}(\zeta_1,\zeta_2), S^A(\zeta_1,\zeta_2)) - \zeta_1 - \zeta_2,$$

when $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ is feasible for Q; otherwise, let $\mathsf{cost}(A,\zeta_1,\zeta_2) = \infty$.

Intuitively, the cost function represents the objective function of the considered subgraph, with artificial weights removed. The artificial weights (ζ_1,ζ_2) are used to model constraints due to non-preemptive regions overlapping with the start or the end of the considered subgraph, i.e., they represent the non-preemptive carry-in and carry-out of the considered subgraph. When computing the WCET of a subgraph in our dynamic programming formulation, such artificial weights will therefore be subtracted.

In the following, we show how to compute the cost function for all valid productions in our grammar $\mathcal{G}$. For each rule, we will provide a cost matrix spanning all meaningful artificial weights: $\zeta_1,\zeta_2 \in \{0,1,\ldots,Q-1\}$. We assume that cost is ∞ for any ζ_1 or ζ_2 that is Q or larger.

For a production (a), the cost matrix can be computed using the following corollary to Theorem 1.

Corollary 1. *When applying production (a) over feasible $G_{\mathcal{P}}$ and Q, the* cost *matrix for the optimal $S^A(\zeta_1,\zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ (where $\zeta_1,\zeta_2 \in \{0,1,\ldots,Q-1\}$) can be constructed by the following recursive computation:*

$$\begin{aligned}&\mathsf{cost}\left(\left[G^A_{\mathcal{P}} \to G^B_{\mathcal{P}}, e_i, G^C_{\mathcal{P}}\right], \zeta_1, \zeta_2\right) \\ &\quad \stackrel{def}{=} \min \left\{ \begin{array}{l} \mathsf{cost}(G^B_{\mathcal{P}}, \zeta_1, 0) + \xi(e_i) + \mathsf{cost}(G^C_{\mathcal{P}}, \xi(e_i), \zeta_2), \\ \min\limits_{x=1}^{Q-1} \left\{\mathsf{cost}(G^B_{\mathcal{P}}, \zeta_1, x) + \mathsf{cost}(G^C_{\mathcal{P}}, Q-x, \zeta_2)\right\} \end{array} \right\}.\end{aligned} \tag{9}$$

For a production (b) deriving either a conditional or a single block ($\langle \texttt{Blocks} \rangle \to \langle \texttt{SB} \rangle | \langle \texttt{CB} \rangle$), the cost function is

$$\begin{aligned}&\mathsf{cost}\left(\left[G^A_{\mathcal{P}} \to (G^{seq}_{\mathcal{P}} \mid G^{con}_{\mathcal{P}})\right], \zeta_1, \zeta_2\right) \\ &\qquad \stackrel{\text{def}}{=} \mathsf{cost}((G^{seq}_{\mathcal{P}} \mid G^{con}_{\mathcal{P}}), \zeta_1, \zeta_2).\end{aligned} \tag{10}$$

For a production (c), instantiating a single basic block ($\langle \texttt{SB} \rangle \to \delta_i$), the cost function is

$$\begin{aligned}&\mathsf{cost}\left([\langle \texttt{SB} \rangle \to \delta_i], \zeta_1, \zeta_2\right) \\ &\qquad \stackrel{\text{def}}{=} \mu_\infty\left(\zeta_1 + \zeta_2 + C(\delta_i) > Q\right) \cdot C(\delta_i).\end{aligned} \tag{11}$$

The following theorem and corollary pertain to the application of production (d), for a conditional block structure $\langle \texttt{CB} \rangle$:

$$\langle \texttt{CB} \rangle \to \delta_i, [e_{i_1}, \langle \texttt{Blocks} \rangle, e_{j_1}], ([e_{i_2}, \langle \texttt{Blocks} \rangle, e_{j_2}]) +, \delta_j.$$

In a general application of production (d), we let $G^A_{\mathcal{P}}$ denote the conditional block on the left-hand side of the rule, and $G^{B_1}_{\mathcal{P}}, \ldots, G^{B_k}_{\mathcal{P}}$ denote the parallel non-terminal blocks on the right-hand side of the rule:

$$G^A_{\mathcal{P}} \to \delta_i, [e_{i_1}, G^{B_1}_{\mathcal{P}}, e_{j_1}], \ldots, [e_{i_k}, G^{B_k}_{\mathcal{P}}, e_{j_k}], \delta_j.$$

where $e_{i_\ell} = (\delta_i, \mathsf{InBlock}(B_\ell))$, and $e_{j_\ell} = (\mathsf{OutBlock}(B_\ell), \delta_j)$ for $\ell = 1,\ldots,k$. Moreover, we let $G^{T_\ell}_{\mathcal{P}}$ denote the subgraph composed of $G^{B_\ell}_{\mathcal{P}}$, and basic blocks δ_i and δ_j as the first and last block, respectively.

Theorem 2. *When applying production (d) over feasible $G_{\mathcal{P}}$ and Q, an optimal set $S^A(\zeta_1,\zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ (where $\zeta_1,\zeta_2 \in \{0,1,\ldots,Q-1\}$) is equal to*

$$\bigcup_{\ell=1}^{k} S^{T_\ell}(\zeta_1,\zeta_2), \tag{12}$$

where for each $\ell \in \{1,\ldots,k\}$, the optimal EPP set $S^{T_\ell}(\zeta_1,\zeta_2)$ equals one of the following sets:

$$S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)), \tag{13}$$

$$S^{B_\ell}(\xi(e_{i_\ell}), \zeta_2 + C(\delta_j)) \cup \{e_{i_\ell}\}, \tag{14}$$

$$S^{B_\ell}(\zeta_1 + C(\delta_i), 0) \cup \{e_{j_\ell}\}, \tag{15}$$

$$S^{B_\ell}(\xi(e_{i_\ell}), 0) \cup \{e_{i_\ell}, e_{j_\ell}\}. \tag{16}$$

Proof: Since the proof is very similar to Theorem 1, a detailed proof is omitted to simplify the reading. The same technique adopted in Case 1 and Case 2 of Theorem 1 is here applied to each parallel subgraph $G^{T_\ell}_{\mathcal{P}}(\zeta_1,\zeta_2)$, for $\ell \in \{1,\ldots,k\}$. For each such subgraph, four different cases are considered:

- $e^\ell_i \notin S^A(\zeta_1,\zeta_2)$, $e^\ell_j \notin S^A(\zeta_1,\zeta_2)$, leading to Eq. (13);
- $e^\ell_i \in S^A(\zeta_1,\zeta_2)$, $e^\ell_j \notin S^A(\zeta_1,\zeta_2)$, leading to Eq. (14);
- $e^\ell_i \notin S^A(\zeta_1,\zeta_2)$, $e^\ell_j \in S^A(\zeta_1,\zeta_2)$, leading to Eq. (15);
- $e^\ell_i \in S^A(\zeta_1,\zeta_2)$, $e^\ell_j \in S^A(\zeta_1,\zeta_2)$, leading to Eq. (16).

The optimal set $S^A(\zeta_1,\zeta_2)$ is simply the union of all subgraphs solutions. ∎

Using the above Theorem, the following method is derived to compute the cost function for production (d).

Corollary 2. *When applying production (d) over feasible $G_{\mathcal{P}}$ and Q, the* cost *matrix for the optimal $S^A(\zeta_1,\zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ (where $\zeta_1,\zeta_2 \in \{0,1,\ldots,m-1\}$) can be constructed by the following recursive computation:*

$$\begin{aligned}
&\mathsf{cost}\left(\left[G^A_{\mathcal{P}} \to \delta_i, [e_{i_1}, G^{B_1}_{\mathcal{P}}, e_{j_1}], \ldots, [e_{i_k}, G^{B_k}_{\mathcal{P}}, e_{j_k}], \delta_j\right], \zeta_1, \zeta_2\right) \\
&\overset{def}{=} \max_{\ell \in \{1,\ldots,k\}} \left\{ \min_{\substack{\alpha\in\{0,1\}\\ \beta\in\{0,1\}}} \left\{ \begin{array}{l} \mathsf{cost}\left(\langle \texttt{SB}\rangle \equiv [\delta_i], \zeta_1, 0\right) \\ \quad +\mu_0(\alpha=0)\cdot\xi(e_{i_\ell}) \\ \quad +\mathsf{cost}\big(G^{B_\ell}_{\mathcal{P}}, \\ \qquad \mu_0(\alpha=1)\cdot(\zeta_1 + C(\delta_i)) \\ \qquad\quad +\mu_0(\alpha=0)\cdot\xi(e_{i_\ell}), \\ \qquad \mu_0(\beta=1)(\zeta_2 + C(\delta_j))\big) \\ \quad +\mu_0(\beta=0)\cdot\xi(e_{j_\ell}) \end{array} \right\} \right\}.
\end{aligned} \tag{17}$$

C. Algorithm Overview

A standard dynamic-programming algorithm follows directly from Corollaries 1 and 2. However, in this subsection, we will give a high-level intuitive overview of the algorithm. The algorithm has two major phases:

Parsing: In this phase, a lexical analyzer will break the input graph into a stream of "tokens". The stream of tokens is used as input into the syntatic analyzer which applies the production rules of Figure 1 to produce a derivation. The derivation for an input graph can be represented by a *parse tree* where each internal node of the tree represents a non-terminal symbol and each leaf is a terminal symbol (e.g., a basic block or edge between a basic block). In the resulting parse tree, a node y is the child of node x when y appears in the left-hand-side of some derivation of x. For instance, Figure 2 can be viewed as a parse tree where the parent-child relation in the parse tree is equivalent to a box being immediately contained inside another box. As an example, the $\langle$SB$\rangle$ node in Figure 2 contains four children: basic blocks C and F (which are also leaf nodes in the parse tree) and two $\langle$Blocks$\rangle$ nodes.

Bottom-up cost **Matrix Computation:** Using the parse-tree as input, our algorithm determines the $Q \times Q$ values of the cost matrix for every node in the parse tree by applying the appropriate computational rules given in the previous subsection. Since the parse tree is traversed in a bottom-up fashion, each internal node can reuse the cost matrix of its children nodes when it is determining its own cost matrix. After the approach above, we can obtain the minimum worst-case execution time for the input graph over all possible sets of EPPs that satisfy Equation (2) by looking at the cost$[0][0]$ entry for the root node of the parse tree.

D. Computational Complexity

The computational complexity of our algorithm is determined by the complexity of generating the parse tree and the complexity of computing the cost matrix for each node. Per the discussion above, the complexity of parse tree is linear in the number of tokens (which is proportional to $|V|$ for the input graph $G_{\mathcal{P}}$). For the complexity of generating the cost matrix for each node of the parse tree, we need to consider the complexity of applying each rule. Given that constructing the rule in Figure 1(b) (i.e., $\langle$Blocks$\rangle \to (\langle$SB$\rangle|\langle$CB$\rangle), e_i, \langle$Blocks$\rangle$) is the most computationally expensive and there are at most $|V|$ nodes in the parse tree, the total time complexity of the implementation described above is $O(|V|Q^3)$. Thus, the runtime of our algorithm is pseudo-polynomial time since it depends upon the value of Q.

§**An Alternative Heuristic.** The above time complexity is potentially still quite large if Q is a large number. In practice, the window of non-preemption (i.e., Q) could be potentially much larger than the basic block size. Thus, it is worthwhile to find an algorithm that does not depend upon the value of Q. In the algorithm described above, for each node A of the parse tree, we compute $Q \times Q$ different solutions for the node. The reason for storing Q^2 values is that at the time we are solving the subproblem corresponding to node A (i.e., $G^A_{\mathcal{P}}$), it has not been determined when the nearest preemption points before or after $G^A_{\mathcal{P}}$ will occur. Thus, we compute all possible combinations. However, it can be shown that using a larger value of preemption points before or after $G^A_{\mathcal{P}}$ will only increase the length of the longest path in $G^A_{\mathcal{P}}$ since more EPPs will need to be selected in the subgraph in order to satisfy the constraint of Equation 2. Thus, as a way to reduce the complexity of the exact approach by potentially obtaining a slight overestimate in the minimum EPP selection, we may fix the dimensions of the cost matrix to be a constant $\alpha \times \alpha \ : \ \alpha \in \mathbb{N}^+$. With this approach, we are only storing values of preemptions before and after each $G^A_{\mathcal{P}}$ equal to

$i \cdot \left\lceil \frac{Q}{\alpha} \right\rceil$ where $i = 0, 1, \ldots, \alpha$. The running time for this heuristic is $O(|V|\alpha^3)$. In the next section, we explore the trade-off between the optimal conditional algorithm and this heuristic.

VII. Evaluation

We have implemented our grammar and algorithm in Java and ANTLRWorks (Version 3) [1]. In this section, we evaluate the performance of our preemption point placement methods over randomly-generated flowgraphs[1].

§**Methodology.** We randomly generate control flowgraphs satisfying the production rules in Figure 1. For our evaluation, we have fixed the number of basic blocks at 400 and the number of "high-level phases" of the flowgraph to be 30. Each high-level phase is either comprised entirely of sequential blocks or is a (non-nested) conditional block with a branching factor of two[2]. For each sequential block (i.e., either a sequential phase or one branch of the conditional), the number of basic blocks is uniformly generated from $[3, 10]$. In our experiments, we vary the number of paths from δ_s to δ_z by increasing the number of conditional blocks; thus, the number of paths is of the form 2^C where C is the number of conditional blocks in the generated control flowgraph.

In Bertogna et al. [8], the authors obtained CRPD and WCET costs for evaluation of their algorithm for sequential control flowgraphs by randomly generating parameters similar to the parameters of realistic code examples (e.g., Simulink code for aircraft and automotive control). In this section, we apply the same methodology to generating CRPD and WCET costs (i.e., $\xi(e_j)$ and $C(\delta_i)$, respectively). WCETs were generated according to a Gaussian distribution with mean equal to 4000 nanoseconds and variance of 3000 nanoseconds. The CRPD for each EPP is generated according to the method in Bertogna et al. [8] that correlates adjacent EPPs; we modify their method to account for conditional branching. Specifically, consider the preemption of $\xi(e_i)$ for EPP e_i:

$$\xi(e_i = (\delta_b, \delta_c)) = \sum_{\delta_a \in \Upsilon_i} \xi(\delta_a, \delta_b)/|\Upsilon_i| + \Delta_i \qquad (18)$$

where $\Upsilon_i \stackrel{\text{def}}{=} \{\delta_a \in V : (\delta_a, \delta_b) \in E\}$, $\Delta_i \stackrel{\text{def}}{=} \mathsf{guas}(m_i, \sigma)$ and

$$m_i \stackrel{\text{def}}{=} \begin{cases} -M & \text{if } \sum_{\delta_a \in \Upsilon_i} \xi(\delta_a, \delta_b)/|\Upsilon_i| > \xi_{\max} \\ +M & \text{if } \sum_{\delta_a \in \Upsilon_i} \xi(\delta_a, \delta_b)/|\Upsilon_i| < \xi_{\min} \\ \mathsf{sgn}(\Delta_{i-1})M & \text{otherwise} \end{cases} \qquad (19)$$

The variance σ quantifies the degree of variability between consecutive EPPs. We use the same parameters as Bertogna et al. [8]: $\sigma = 3000$, $M = 20$, $\xi_{\min} = 1000$, and $\xi_{\max} = 55000$.

In our experiments, we compare the optimal algorithm proposed in this paper described in Section VI-D (denoted as "COND(OPT)") with an application of the sequential algorithm of Bertogna et al. [8] (denoted as "SEQ"). SEQ consists of generating each path in the conditional control flowgraph and applying the previously-proposed sequential algorithm to determine the EPPs along each path. The EPPs of each path are unioned together to obtain the overall EPP selection to conditional flowgraph. We also compare these algorithms with the heuristic proposed in Section VI-D (denoted as "COND($\alpha \times \alpha$)" where a value of α equals 50, 100, 500). We execute the algorithms on a 2.4GHz 4-core Intel $i7$ machine with 6GB of RAM.

[1]Bertogna et al. [8] provides justification of the parameter selection.

[2]We do not nest in order to control the number of paths from δ_s to δ_z. We have observed that the results for nested conditional blocks are similar to the ones presented here.

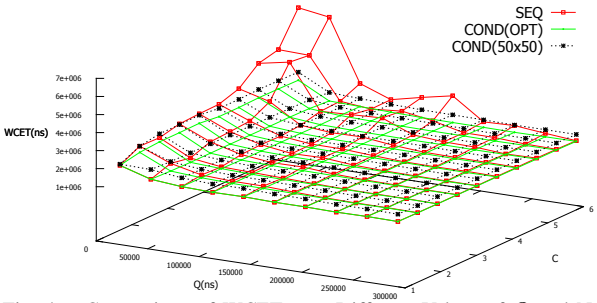


Fig. 4. Comparison of WCET over Different Values of Q and Number of Conditional Blocks (C) for SEQ, COND(OPT), and COND(50×50).

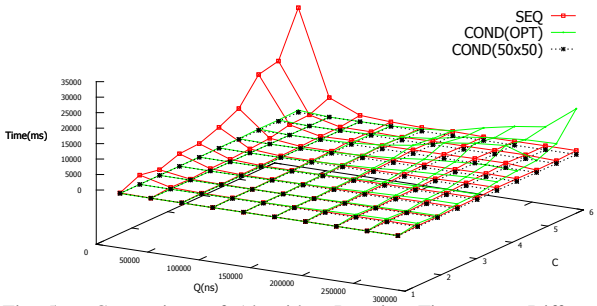


Fig. 5. Comparison of Algorithm Running Times over Different Values of Q and Number of Conditional Blocks (C) for SEQ, COND(OPT), and COND(50×50)

We vary the non-preemption window (denoted by Q in nanoseconds) and the number of conditional blocks in the graph (denoted by C, resulting in 2^C total paths from δ_s to δ_z). For each (Q, C) pair, we use the above methodology to generate 100 different control flowgraphs. For each control flowgraph, we apply both algorithms and measure the resulting WCET and running time of the algorithm. The averages of the WCETs are plotted in Figures 4 and 6, and the average running algorithm running times in Figures 5 and 7.

§**Results.** Since COND(OPT) has been shown to be optimal for flowgraphs considered in this paper, it is not surprising that Figure 4 shows that WCET of COND(OPT) never exceeds the

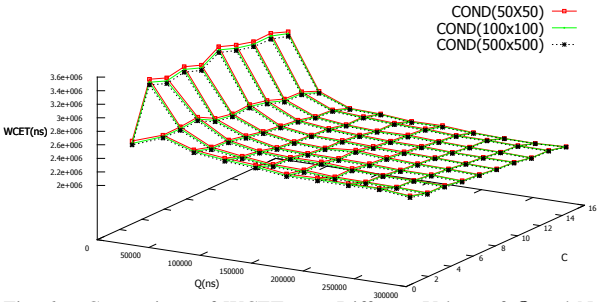


Fig. 6. Comparison of WCET over Different Values of Q and Number of Conditional Blocks (C) for heuristics.

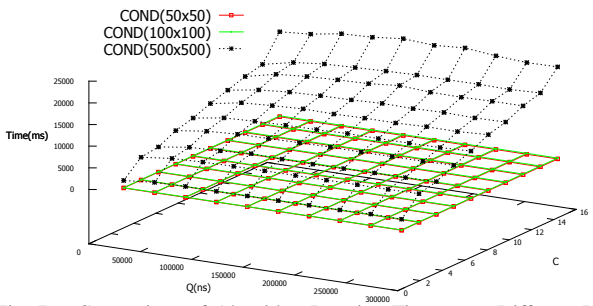


Fig. 7. Comparison of Algorithm Running Times over Different Values of Q and Number of Conditional Blocks ((C) for heuristics.

WCET returned by SEQ or the heuristic COND(50×50). In Figure 5, as Q decreases, the preemption cost becomes a larger factor in the WCET since there are more preemptions due to the small non-preemptive window. As C increases, SEQ will add an increasingly large number of unnecessary EPPs to the solution set since it does not consider the conditional structure. These two observations explain the sharp increase in WCET for the sequential approach for small Q and C approaching a value of 6 (i.e., 2^6 total paths). After 2^6 paths, COND(OPT) memory requirements grows quickly for large Q, and the runtime of SEQ sharply increases. These increases make it difficult to scale SEQ or COND(OPT) to larger values.

In Figures 4 and 5, we observe that the heuristic COND(50×50) returns WCET close to the optimal with small running time when compared with SEQ and COND(OPT). We further compare the various heuristics in terms of the accuracy/running-time tradeoff in Figures 6 and 7. These figures show that the accuracy does not change much for α equal to 50, 100, or 500 for values of C up to 15 (i.e, there are up to 2^{15} paths in the graphs). However, for $\alpha = 500$, the memory costs of the `cost` matrices begin to dominate and increase the running time as the parse tree grows more complicated with increasing C. Thus, we can obtain a better accuracy-to-cost benefit with a relatively small value of α equal to 50.

VIII. Conclusions

In this paper, we extended the applicability of existing techniques for the placement of preemption points to general tasks modeled with control flowgraphs, removing a pessimistic assumption that conditional blocks are contained within basic blocks. The proposed method allows optimal selection of EPPs that minimize the resulting worst-case execution time, without affecting the schedulability of the system. The method uses a combination of graph grammars and dynamic programming, and runs in pseudo-polynomial time. Our evaluation shows that the algorithm can achieve a reduction in WCET over approaches handling only sequential control flowgraphs. The improvement is particularly important when the allowed non-preemptive region length (Q) is small (i.e., for heavily loaded systems with limited slack). Such a system configuration is favorable also for the running time of the algorithm, which is $O(Q^3)$. We have also proposed heuristics which have observed near-optimal behavior with very low runtime cost and have addressed important difficult-to-handle substructures such as non-unrolled loops and non-inline functions in the appendix. As a future work, it would be interesting to see how the proposed approach could be directly integrated into automatic programming/compiler tools.

References

[1] Antlrworks: The antlr gui development environment. http://www.antlr3.org/works/.

[2] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[3] S. Altmeyer, R. Davis, and C. Maiza. Improved cache related preemption delay aware response time analysis for fixed-priority systems. *Real-Time Systems*, 48(5):499–512, September 2012.

[4] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proc. of the 17th Euromicro Conf. on Real-Time Systems (ECRTS'05)*, pages 137–144, Palma de Mallorca, Balearic Islands, Spain, July 6-8, 2005.

[5] M. Bertogna and S. Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 6(4):579–591, 2010.

[6] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo. Preemption points placement for sporadic task sets. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS'10)*, Brussels, Belgium, June 2010.

[7] M. Bertogna, G. Buttazzo, and G. Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *Proceedings of 32nd IEEE Real-Time Systems Symposium (RTSS 2011)*, Vienna, Austria, Nov. 30 - Dec. 2, 2011.

[8] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *Proceedings of the Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2011. IEEE Computer Society Press.

[9] C. Böhm and G. Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, May 1966.

[10] R. Bril, J. Lukkien, and W. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time System*, 42(1-3):63–119, 2009.

[11] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. *S. Son, editor, Advances in Real-Time Systems*, pages 225–248, 1994.

[12] G. Buttazzo, M. Bertogna, and G.Yao. Limited preemptive scheduling for real-time systems: a survey. *IEEE Transactions on Industrial Informatics*, 9(1), 2013.

[13] R. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred preemption. In *Real-Time Systems Symposium (RTSS 2012)*, San Juan, Puerto Rico, December 4-7, 2012.

[14] K. Kennedy and L. Zucconi. Applications of a graph grammar for program control flow analysis. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 72–85, New York, NY, USA, 1977. ACM.

[15] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *IEEE Real-Time Systems Symposium*, pages 212–224, Rio de Janeiro, Brazil, 2006.

[16] M. L. Scott. *Programming Language Pragmatics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2009.

[17] R. Wilhelm et al. The worst-case execution-time problemoverview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, 2008.

[18] Y. Y. X. He. Parallel recognition and decomposition of two terminal series parallel graphs. *Information and Computation*, 75(1):15–38, 1987.

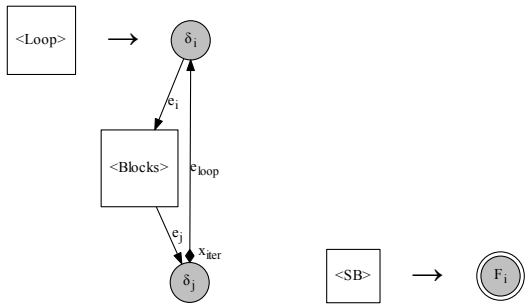
[19] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *Proc. of the 15th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009)*, Beijing, China, August 24-26, 2009.

APPENDIX

A. Practical Extensions: Non-Inline Functions & Non-Unrolled Loops

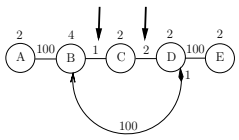
Up until this point in the paper, we have only considered series-parallel flowgraphs which preclude important program flow structures such as non-inline functions and non-unrolled loops. The reasons for not unrolling loops and not inlining functions are sound: i) the program size will be decreased; and ii) subsequent invocations of the same instruction may be in the instruction cache, decreasing the CRPD. Therefore, it is important to be able to address programs with these substructures. However, we cannot directly apply the dynamic programming algorithm presented in Section VI since EPP selection cannot change in successive invocations of a loop iteration or a function; this is due to the fact that there is only a single copy of the code and the EPP code must be placed within this reusable segment of code and cannot change locations from invocation to invocation. Contrast this to using our dynamic-programming algorithm on unrolled loops and/or inline functions; each invocation could potentially have a completely different placement of EPPs since we have made multiple copies of the invocation flowgraph structure in our program.

Figure 8 gives the added productions for handling non-unrolled loop structures (Figure 8(a)) and non-inline functions (Figure 8(b)). For this paper, due to space constraints, we provide the loop production for do...while() loop structures only; other loop structures can be handled with similar production rules and the method presented in this appendix only needs slight modification for these other structures. We assume that each loop structure has a bound (x_{iter}) on the number of times it branches back to the beginning of the loop. As an example of the differences in unrolled versus non-unrolled structures consider Figure 9 with a loop that repeats at most once for $Q = 8$. In the non-unrolled-loop example in Figure 9(a), both EPPs within the loop must be selected since only selecting one is not feasible; the resulting WCET+CRPD

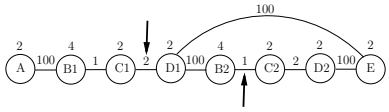


(a) $\langle \texttt{Loop} \rangle \rightarrow \{\delta_i, e_i, \langle \texttt{Blocks} \rangle, e_j, \delta_j, e_{\text{loop}}, x_{\text{iter}}\}$ (b) $\langle \texttt{SB} \rangle \rightarrow F_i$

Fig. 8. Additional production rules to handle unrolled loops and non-inline functions.



(a) Non-Unrolled Loop Example



(b) Unrolled Loop Example

Fig. 9. Non-Unrolled Loop vs. Unrolled Loop

for this selection is 26. In the corresponding unrolled-loop example in Figure 9(a), we may select the more costly edge on the first loop iteration and the less costly on the next iteration; the resulting WCET+CRPD for this case is 23.

§**Non-Unrolled Loops.** To handle non-unrolled loops, we can derive a recursive formulation similar to the ones presented in Section VI. That is, we can represent the optimal set of EPPs for any loop subgraph in terms of the optimal solution of its contingent $\langle \texttt{Blocks} \rangle$ subgraph. In the following theorem, we show the optimal substructure for a $\langle \texttt{Loop} \rangle$. In a general application of production of Figure 8(a), we let $G^A_{\mathcal{P}}$ denote the non-terminal $\langle \texttt{Loop} \rangle$ block on the left-hand side of the rule, and $G^B_{\mathcal{P}}$ denote the non-terminal blocks in the loop iteration:

$$G^A_{\mathcal{P}} \rightarrow \{\delta_i, e_i, \langle \texttt{Blocks} \rangle, e_j, \delta_j, e_{\text{loop}}, x_{\text{iter}}\}$$

where $e_i = (\delta_i, \mathsf{InBlock}(G^B_{\mathcal{P}}))$ and $e_j = (\mathsf{OutBlock}(G^B_{\mathcal{P}}), \delta_j)$.

Theorem 3. *When applying production in Figure 8(a) over feasible $G_{\mathcal{P}}$ and Q, an optimal set $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$) is equal to one of the following:*

$$S^B(\xi(e_i), 0) \cup \{e_i, e_j, e_{\text{loop}}\}, \tag{20}$$

$$S^B(\xi(e_i), 0) \cup \{e_i, e_j\}, \tag{21}$$

$$S^B(\max(\zeta_1 + c(\delta_i), \xi(e_{\text{loop}}) + c(\delta_i)), 0) \cup \{e_j, e_{\text{loop}}\}, \tag{22}$$

$$S^B(\xi(e_i), \max(c(\delta_i) + c(\delta_j), c(\delta_j) + \zeta_2)) \cup \{e_i\}, \tag{23}$$

$$S^B(\max(\zeta_1 + c(\delta_i), \xi(e_{\text{loop}}) + c(\delta_i)), c(\delta_j) + \zeta_2) \cup \{e_{\text{loop}}\}, \tag{24}$$

$$S^B(\xi(e_i), c(\delta_j) + \zeta_2) \cup \{e_i, e_{\text{loop}}\}, \tag{25}$$

$$S^B(\max(\zeta_1 + c(\delta_i), \xi(e_j) + c(\delta_j) + c(\delta_i)), 0) \cup \{e_j\}, \tag{26}$$

or one of the following for $a \in \{0, \ldots, Q - \zeta_1 - c(\delta_i)\}$ and $b \in \{0, \ldots, Q - \zeta_2 - c(\delta_j)\}$ when $Q \leq \zeta_1 + \zeta_2 + a + b$:

$$S^B(\zeta_1 + c(\delta_i) + a, \zeta_2 + c(\delta_j) + b). \tag{27}$$

Proof Sketch: The proof is very similar to Theorem 1 but extremely long due to the number of cases and the length of the proof for each case. There are eight cases depending upon whether edges e_i, e_j, or e_{loop} are selected as EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$. The sets of Equations 20-26 correspond to subproblems where at least one of e_i, e_j, or e_{loop} has been selected; the sets represented in Equation 27 correspond to where none of the edges e_i, e_j, and e_{loop} is selected. To give

some intuition, we give an informal description of the proof for Equation 24 and Equation 27 only; Equations 20-26 are quite similar to Equation 24.

Equation 24 corresponds to the scenario that only e_{loop} is selected among the set $\{e_i, e_j, e_{\text{loop}}\}$. Thus, using the right-hand-side of the production in Figure 8(a), we must reason about the possible distance of preemption points before and after the non-terminal node $\langle\texttt{Blocks}\rangle$ represented by the subgraph $G^B_{\mathcal{P}}$. In the first iteration, the last preemption before $\mathsf{InBlock}(G^B_{\mathcal{P}})$ occurs $\zeta_1 + c(\delta_i)$ time units prior, by construction of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$. However, in subsequent iterations of the loop, the last preemption before $\mathsf{InBlock}(G^B_{\mathcal{P}})$ is $\xi(e_{\text{loop}}) + c(\delta_i)$ units prior. Thus, when we determine the set of EPPs selected from $G^B_{\mathcal{P}}$, we must assume the worst-case that the preemption is $\max(\zeta_1 + c(\delta_i), \xi(e_{\text{loop}}) + c(\delta_i))$ units prior to $\mathsf{InBlock}(G^B_{\mathcal{P}})$ for ensuring the constraint of Equation 2 is not violated. Similarly, for the first preemption after $\mathsf{OutBlock}(G^B_{\mathcal{P}})$, the last iterations has it $\zeta_2 + c(\delta_j)$ after $\mathsf{OutBlock}(G^B_{\mathcal{P}})$. In previous iterations, it is $c(\delta_j)$ units prior to $\mathsf{OutBlock}(G^B_{\mathcal{P}})$ (which is dominated by the previous case). Therefore, if only e_{loop} is selected, the set $S^B(\max(\zeta_1 + c(\delta_i), \xi(e_{\text{loop}}) + c(\delta_i)), c(\delta_j) + \zeta_2)$ gives us the minimum cost for $G^A_{\mathcal{P}}$ in this case.

Equation 27 corresponds to the scenario that no edge is selected among the set $\{e_i, e_j, e_{\text{loop}}\}$. In this case, the first iteration of the loop has the last preemption occurring $\zeta_1 + c(\delta_i)$ prior to $\mathsf{InBlock}(G^B_{\mathcal{P}})$; the last iteration of the loop has first preemption occurring $\zeta_2 + c(\delta_i)$ after $\mathsf{OutBlock}(G^B_{\mathcal{P}})$. The set $S^B(\zeta_1 + c(\delta_i), \zeta_2 + c(\delta_j))$ will ensure the EPPs satisfy the constraints of Equation 2 with respect to these preemptions before and after $G^B_{\mathcal{P}}$. However, the EPPs selected in this set may potentially violate Equation 2 between iterations of the loop; i.e., the time between last preemption in $S^B(\zeta_1 + c(\delta_i), \zeta_2 + c(\delta_j))$ and the first preemption in the same set (traveling through BBs δ_j and δ_i) could be larger than Q. By construction, the last preemption in $S^B(\zeta_1 + c(\delta_i), \zeta_2 + c(\delta_j))$ must occur within $Q - \zeta_2 - c(\delta_j)$ before the end of $G^B_{\mathcal{P}}$; the first preemption must occur $Q - \zeta_1 - c(\delta_i)$. The total time between preemptions in $G^B_{\mathcal{P}}$ for this set may be as large as $(Q - \zeta_1 - c(\delta_i)) + (Q - \zeta_2 - c(\delta_j)) + c(\delta_j) + c(\delta_i)$. To force the preemptions to be closer, we may show that we can add a non-negative a and b value to the preemption times before and after $G^A_{\mathcal{P}}$; adding these values gives a distance of $2Q - \zeta_1 - \zeta_2 - a - b$ between preemptions. To satisfy Equation 2 and have the distance between preemptions at most Q, we must have $Q \leq \zeta_1 + \zeta_2 + a + b$. Thus, for this case we obtain the minimum solution from the sets $S^B(\zeta_1 + c(\delta_i) + a, \zeta_2 + c(\delta_j) + b)$ for all a and b that satisfy the constraints set in the statement of the theorem. □

The following corollary follows immediately from Theorem 3 by calculating the sets and checking the EPP constraint (Equation 2) for the basic blocks of the loop.

Corollary 3. *When applying production of Figure 8(a) over feasible $G_{\mathcal{P}}$ and Q, the* cost *matrix for the optimal $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$) can be constructed taking the minimum of Equation 28 in Figure 10 (over all values of $(\alpha, \beta, \gamma) \in \{0,1\}^3$ where at least one value is one) and Equation 29 in Figure 11. The time complexity for each pair (ζ_1, ζ_2) is $O(Q^2)$.*

§**Non-Inline Functions.** To address non-inline functions, we suggest a simple straightforward approach. Let G_F represent the control flowgraph for any function F. Before calculating the WCET+CRPD for program $\mathcal{P}$, we first calculate the cost matrix of G_F for all $\zeta_1, \zeta_2 \in \{0, \ldots, Q-1\}$. Then, for each invocation of function F in the program, we will fix ζ_1 and ζ_2 for F and create a cost matrix to use for F (i.e., $\mathsf{cost}[F, \cdot, \cdot]$) when applying the dynamic programming algorithm for $G_{\mathcal{P}}$. For any non-negative $x \leq \zeta_1$ and $y \leq \zeta_2$, $\mathsf{cost}[F, x, y]$ will be $\mathsf{cost}[G_F, \zeta_1, \zeta_2]$; otherwise, if $x > \zeta_1$ or $y > \zeta_2$, $\mathsf{cost}[F, x, y]$ is ∞. Intuitively, we are statically fixing the EPPs of F and forcing the preemptions to be within ζ_1 and ζ_2 before and after each invocation of F. We can iteratively try each F for all Q^2 values of ζ_1 and ζ_2. Thus, if there are K distinct non-inline function this approach will try Q^{2K} combinations. We are searching for a less computationally-intensive approach.

$$
\begin{aligned}
(x_{\text{iter}} + 1) \cdot \Bigg[& \; \mu_\infty\left((\zeta_1 + c(\delta_i) > Q) \vee (c(\delta_i) + \mu_0(\gamma = 0) \cdot \xi(e_{\text{loop}}) + \mu_0\left((\gamma = 1) \vee (\beta = 0)\right) \cdot \xi(e_j) + \mu_0(\gamma = 1) \cdot c(\delta_j) > Q)\right) \cdot c(\delta_i) \\
& + \mu_0(\alpha = 0) \cdot \xi(e_i) \\
& + \mathsf{cost}\left[\begin{array}{l} G^B_{\mathcal{P}}, \\ \mu_0(\alpha = 0) \cdot \xi(e_i) + \mu_0(\alpha = 1) \cdot [c(\delta_i) + \max\left(\zeta_1, \mu_0(\gamma = 0) \cdot \xi(e_{\text{loop}}) + \mu_0(\gamma = 1) \cdot (\xi(e_j) + c(\delta_j))\right)], \\ \mu_0(\beta = 1) \cdot \max\left(c(\delta_j) + \zeta_2, c(\delta_j) + \mu_0(\gamma = 1) \cdot c(\delta_i)\right) \end{array}\right) \\
& + \mu_0(\beta = 0) \cdot \xi(e_j) \\
& + \mu_\infty\left((\mu_0(\beta = 0) \cdot \xi(e_j) + c(\delta_j) + \zeta_2 > Q) \vee (\mu_0(\beta = 0) \cdot \xi(e_j) + c(\delta_j) + \mu_0(\gamma = 1) \cdot c(\delta_i) > Q)\right) \cdot c(\delta_j) \\
\Bigg] & \\
+ \mu_0(\gamma = 0) \cdot x_{\text{iter}} \cdot \xi(e_{\text{loop}}).
\end{aligned}
\tag{28}
$$

Fig. 10. Equation that calculates the value of the subproblems for a $\langle\texttt{Loop}\rangle$ production corresponding to Equations 20-26 in Theorem 3. Here $\alpha, \beta, \gamma \in \{0, 1\}$ are boolean values representing whether edges e_i, e_j, or e_{loop} are among the selected EPPs.

$$
\begin{aligned}
(x_{\text{iter}} + 1) \cdot \Bigg[& \; \mu_\infty\left((\zeta_1 + c(\delta_i) > Q) \vee (c(\delta_i) + c(\delta_j) > Q)\right) \cdot c(\delta_i) \\
& + \min_{\substack{0 \leq a \leq Q - \zeta_1 - c(\delta_i) \\ 0 \leq b \leq Q - \zeta_2 - c(\delta_j)}} \left\{\mu_\infty(Q > \zeta_1 + \zeta_2 + a + b) \cdot \mathsf{cost}(G^B_{\mathcal{P}}, \zeta_1 + c(\delta_i) + a, \zeta_2 + c(\delta_j) + b)\right\} \\
& + \mu_\infty\left((c(\delta_j) + \zeta_2 > Q) \vee (c(\delta_j) + c(\delta_i) > Q)\right) \cdot c(\delta_j) \\
\Bigg] &
\end{aligned}
\tag{29}
$$

Fig. 11. Equation that calculates the value of the subproblem for a $\langle\texttt{Loop}\rangle$ production corresponding to Equation 27 in Theorem 3.