On Systematic Testing for Execution-Time Analysis

Daniel Bundala
UC Berkeley
Email: bundala@berkeley.edu

Sanjit A. Seshia
UC Berkeley
Email: sseshia@berkeley.edu

Abstract—Given a program and a time deadline, does the program finish before the deadline when executed on a given platform? With the requirement to produce a test case when such a violation can occur, we refer to this problem as the worst-case execution-time testing (WCETT) problem.

In this paper, we present an approach for solving the WCETT problem for loop-free programs by timing the execution of a program on a small number of carefully calculated inputs. We then create a sequence of integer linear programs the solutions of which encode the best timing model consistent with the measurements. By solving the programs we can find the worst-case input as well as estimate execution time of any other input. Our solution is more accurate than previous approaches and, unlike previous work, by increasing the number of measurements we can produce WCETT bounds up to any desired accuracy.

Timing of a program depends on the properties of the platform it executes on. We further show how our approach can be used to quantify the timing repeatability of the underlying platform.

I. INTRODUCTION

Execution-time analysis is central to the design and verification of real-time embedded systems. In particular, over the last few decades, much work has been done on estimating the worst-case execution time (WCET) (see, e.g. [1], [2], [3]). Most of the work on this topic has centered on techniques for finding upper and lower bounds on the execution time of programs on particular platforms. Execution time analysis is a challenging problem due to the interaction between the interaction of a large space of program paths with the complexity of underlying platform (see, e.g., [4], [5]). Thus, WCET estimates can sometimes be either too pessimistic (due to conservative platform modeling) or too optimistic (due to unmodeled features of the platform).

The above challenges for WCET analysis can limit its applicability in certain settings. One such problem is to verify, given a program \( P \), a target platform \( H \), and a deadline \( d \), whether \( P \) can violate deadline \( d \) when executed on \( H \) — with the requirement to produce a test case when such a violation can occur. We refer to this problem as the worst-case execution-time testing (WCETT) problem.

Tools that compute conservative upper bounds on execution time have two limitations in addressing this WCETT problem: (i) if the bound is bigger than \( d \), one does not know whether the bound is too loose or whether \( P \) can really violate \( d \), and (ii) such tools typically aggregate states for efficiency and hence do not produce counterexamples. Moreover, such tools rely on having a fairly detailed timing model of the hardware platform (processor, memory hierarchy, etc.). In some industrial settings, due to IP issues, hardware details are not readily available, making the task much harder for timing analysis (see e.g. this NASA report for more details [6]); in such settings, one needs an approach to timing analysis that can work with a “black-box” platform.

In this paper, we present an approach to systematically test a program’s timing behavior on a given hardware platform. Our approach can be used to solve the WCETT problem, in that it not only predicts the execution times of the worst-case (longest) program path, but also produces a suitable test case. It can also be adapted to produce the top \( K \) longest paths for any given \( K \). Additionally, the timing estimate for a program path comes with a “guard band” or “error bound” characterizing the approximation in the estimate — in all our experiments, true value was closed to the estimate and well within the guard band.

Our approach builds upon on prior work on the GameTime system [7], [8]. GameTime allows one to predict the execution time of any program path without running it by measuring a small sample of “basis paths” and learning a timing model of the platform based on those measurements.

The advantage of the GameTime approach is that the platform timing model is automatically learned from end-to-end path measurements, and thus it is easy to apply to any platform. However, the accuracy of GameTime’s estimate (the guard bands) depend on the knowledge of a platform parameter \( \mu_{\text{max}} \) which bounds the cumulative variation in the timing of instructions along a program path from a baseline value.

For example, a load instruction might take just 1 cycle if the value is in a register, but several 10s of cycles if it is in main memory and not cached. The parameter \( \mu_{\text{max}} \) can be hard to pre-compute based on documentation of the processors ISA or even its implementation, if available.

Our approach shares GameTime’s ease of portability to any platform, and like it, is also suitable for black-box platforms. However, rather than depending on knowledge of \( \mu_{\text{max}} \), we show how one can compute the guard bands using an integer linear programming formulation. Experimental results show that our approach can be more accurate than the original GameTime algorithm [8], at a small extra computational cost. Moreover, our algorithm is tunable: depending on the desired accuracy specified by a user, the algorithm measures more paths and yields more precise estimate; possibly measuring all paths if perfect accuracy is requested. Finally, we also show how to estimate the parameter \( \mu_{\text{max}} \).
Our solution to the problem is an extension of the measurement-based GameTime approach [11]. We now present the model used in [11] as well as in this paper.

A. Model

To decide whether a program can exceed a given time limit \( d \), it suffices to decide whether the worst input exceeds the limit \( d \). However, recall that without any restrictions on the program, when the program contains unbounded loops or recursion, even determining whether the program terminates not even the number of steps it performs is undecidable. Therefore, we consider only deterministic programs with bounded loops and no recursion. We did not find this limitation restricting as reactive controllers are already often written with this limitation in mind.

Given a computer program, one can associate with it the control-flow graph (CFG) in the usual way (Figure 1); vertices representing locations and edges the basic blocks (straight-line code fragments) of the program. Since we assume that the loops are bounded and there is no recursion, the loops can be unrolled and all function calls inlined. Thus, the resulting CFG is always a directed and an acyclic graph (DAG).

Given a DAG \( G = (V, E) \), with the set of vertices \( V \), set of edges \( E \), we designate two vertices: source \( s \) and sink \( t \) corresponding to the entry and exit points of the program, respectively.

For vertex \( v \) of graph, we use \( \text{in}(v) \) to denote the set of incoming edges to \( v \) and \( \text{out}(v) \) to denote the set of outgoing edges from \( v \).

To model [11] the execution times of a program, we associate with each edge \( e \in E \) cost \( w_e \). The cost \( w_e \) models the (baseline) execution time of the particular statement the edge \( e \) corresponds to.

As described in the introduction, we measure only execution times of entire source-to-sink paths and not of individual statements. Given a source-to-sink path \( x \), the baseline execution time of the path \( x \) is \( w_x = \sum_{e \in x} w_e \) where the sum is over all edges present in \( x \). However, due to caching, branch misses, etc. the actual execution time differs from the baseline and thus the execution time (the length of the path) can be modeled as

\[
\begin{align*}
w_x &= \sum_{e \in x} w_e + d_x \\
\end{align*}
\]

where the term \( d_x \) denotes the variation from the baseline value. The term \( d_x \) is a function of the input and the context the program runs in. This is known as the bounded path-dependent repeatability condition [12].

The length of path \( x \) is denoted \( w_x \). Observe that different inputs corresponding to the same path in general take different time to execute. However, we assume that \( |d| \leq \mu_x \) for some \( \mu_x \in \mathbb{R} \). We denote the maximum \( \max_x \mu_x \) by \( \mu_{\max} \). The value \( \mu_{\max} \) is a measure of timing repeatability. If \( \mu_{\max} = 0 \) then the system is perfectly time repeatable. In general, the larger the value of \( \mu_{\max} \), the less repeatable the timing of the system is.

The aim of this paper is to find a path \( x \) such that \( w_x \) is maximal. The algorithm in [11] as well as ours, does not require the knowledge of \( \mu_x \)'s or \( \mu_{\max} \) to find the worst-case execution input. However, the accuracy of the algorithms depend of \( \mu_{\max} \), as that is inherent to the timing of the underlying hardware.

Our algorithm carefully synthesizes a collection of inputs on which it runs the given program and measures the times it takes to execute each of them. Then, using these measurements, it estimates the length of the longest path.

Formally, the pair \( (x, w_x) \) consisting of a path \( x \) and its length \( w_x \) is called a measurement. We denote the length of the longest path by \( w_M \).

To summarize, throughout the paper we use the following notation.

- \( G \) - underlying DAG
- \( S \) - set of measured paths
- \( M = \{(x_i, l_i)\} \) - set of measurements consisting of pairs path \( x_i \in S \) and the observed length \( l_i \)

It was shown in [11] how, using only \( |E| \) measurements of source-to-sink paths, to find an input corresponding to a path of length at least \( w_M = 2|E|\mu_{\max} \). In particular, if the longest path is longer than the second longest path by at least \( 2|E|\mu_{\max} \), the algorithm in [11] in fact finds the longest path. Thus, we say that the “accuracy” of the algorithm is \( 2|E|\mu_{\max} \).

\begin{verbatim}
int f(int x) {
    if (x % 2 == 0) {
        if (x & 101) {
            x++;  
        } else {
            x+=7;  
        }
    } else {
        x += 7;
    }
    return x;
}
\end{verbatim}

Fig. 1: Source code (top) and its corresponding control flow graph (bottom)
In this paper, we show how to (i) improve the accuracy without increasing the number of measurements, (ii) by increasing the number of measurements improve the accuracy even further, (iii) how to estimate the timing repeatability of the underlying platform (as captured by \( \mu_{\text{max}} \)).

Our algorithm as well as the algorithm in [11] measures the length of some paths and then estimates the lengths of other paths based on those measurements. Note that as long as not all the lengths of all the paths are measured, the inaccuracy of estimates is unavoidable. Consider for example the graph in Figure 2 and assume the graph consists of 6 "diamonds". Clearly, there are 2^6 source-to-sink paths in the graph.

Assume that \( w_e = 1 \) for each edge and \( \mu_e = 0 \) for all paths \( x \) except for one path \( y \) for which \( \mu_y = \mu_{\text{max}} > 0 \). Now, suppose we measure the lengths of some collection of paths \( S \). As long as \( S \) does not contain \( y \), the length of every observed path is 2n. Hence, any length of \( w_y \) of \( y \) in the interval \([2n - \mu_{\text{max}}, 2n + \mu_{\text{max}}]\) is consistent with the measurement. Therefore, in the worst case, the best achievable estimate of the length of \( w_y \) can always be at least \( \mu_{\text{max}} \) from the real answer.

We now briefly describe the algorithm in [11]; we skip the standard technical details such as CFG extraction or how to find an input corresponding to a given path and focus only on how to extract the longest path.

Let \( m \) be the number of edges in \( E \). Then, by numbering the edges in \( E \), one can think of each path \( x \) as a vector \( p_x \) in \( \mathbb{R}^m \) such that

\[
p_x(i) = \begin{cases} 
0 & \text{if ith edge is not used in } x \\
1 & \text{if ith edge is used in } x 
\end{cases}
\]

Now, given two paths \( x \) and \( y \), one can define the linear combination \( a \cdot x + b \cdot y \) for \( a, b \in \mathbb{R} \) in the natural (component wise) way. Thus, one can think of paths as points in an \( m \)-dimensional vector space over \( \mathbb{R} \). In particular, it was show in [11] that there is always a basis \( B \) of at most \( m \) source-to-sink paths such that each source-to-sink path \( x \) can be written as a linear combination of paths from \( B \):

\[
p_x = \sum_{b \in B} c_b \cdot p_b
\]

where \( c_b \)'s are coefficients in \( \mathbb{R} \). Moreover, using theory of 2-barycentric spanners [13], it was shown that \( B \) can be chosen in such a way that for any path \( x \), it always holds that \( |c_b| \leq 2 \) for every \( b \in B \). The paths in \( B \) are called the basis paths as they suffice to express any other path.

Now, if the path \( p_x \) can be written as \( p_x = \sum_{b \in B} c_b \cdot p_b \) then its estimated (baseline) length is

\[
w_x = \sum_{b \in B} c_b \cdot w_b
\]

where \( w_b \)'s are the measured lengths of the basis paths.

The algorithm thus runs the program on the inputs corresponding to the basis paths in order to measure the lengths \( p_b \) for each \( b \in B \). Moreover, it was shown in [11] how, by encoding the problem as an integer-linear-program (ILP) instance, to find path \( X \) such that the corresponding estimated length: \( p_X = \sum_{b \in B} c_b \cdot w_b \) is maximized.

Consider the accuracy of the estimated length of \( p_X \). By construction, \( p_X = \sum_{b \in B} c_b \cdot p_b \). Hence,

\[
\sum_{e \in X} w_e = \sum_{b \in B} c_b \sum_{e \in b} w_e.
\]

Further, for \( b \in B \), we have \( w_b = \sum_{e \in b} w_e + d_b \). Hence,

\[
w_X - \sum_{b \in B} c_b \cdot w_b = \left| \sum_{e \in X} w_e + d_X - \sum_{b \in B} c_b \cdot w_b \right| = \left| \sum_{e \in X} w_e + d_X - \sum_{b \in B} c_b \left( \sum_{e \in b} w_e \right) \right| = \left| \sum_{e \in X} w_e - \sum_{b \in B} c_b \sum_{e \in b} w_e + d_X - \sum_{b \in B} c_b \cdot d_b \right| \leq \left| d_X - \sum_{b \in B} c_b \cdot d_b \right| \leq (2|B| + 1)\mu_{\text{max}}
\]

Thus, the algorithm in [11] finds the longest path (and the corresponding input) only up to the error term of \((2|B| + 1)\mu_{\text{max}}\), under certain assumptions outlined in [11]. A challenge, as noted earlier, is that it is not easy to estimate the value of \( \mu_{\text{max}} \). Consider the first four columns in Table 7. The second column in the table shows the lengths of the longest path as estimated by the algorithm in [11]. However, the third column shows the actual length of the path that is measured when the program is executed on the corresponding inputs. Notice that in some cases the prediction does not match the measured time. Also, the algorithm in [11] does not provide any error bounds on the estimates, thereby making the predicted values less useful.

In this paper we show how, given the exactly same set of measurements as in [11], we can find a tighter estimate and how to incorporate the knowledge of additional measurements to obtain even tighter bounds. In fact, for the benchmarks given

\[\text{In general, however, it holds that the more paths are included in } S \text{ the better is the estimate of the longest path.}\]
in Table V we not only obtain more accurate predictions of running time, we can also give error bounds for our estimates.

III. ALGORITHM

A. Overview

We now give an overview of our algorithm. Recall, that the problem studied can be considered as follows: Given a DAG with source $s$ and sink $t$, find the longest source-to-sink path where the lengths of that paths are modeled as described in the Preliminaries, Section II.

The algorithm in [11] expresses every path as a linear combination of basis paths; using their lengths to estimate the length of the paths not measured. Intuitively, if two paths overlap (they share common edges) then knowing the length of one provides some information about the length of the other. Even basis paths with zero coefficient in the linear combination can provide information about the length of an estimated path.

In our algorithm, we write integer linear programs (ILPs), with one constraint per measurement, looking for the longest path with the edge weights consistent with the measurements and $\mu_x$. Even though, $\mu_x$ are not observable, we show how to obtain consistent bound on $\mu_{\text{max}}$ from the measurements.

B. Path Extraction

In this section we assume that we have a set of measurements $M$, consisting of pairs $(x, l_x)$ where $x$ is a path and $l_x$ is the measured length of $x$ and we show how to find the longest path consistent with the measurements. In Section III-D we then show how to actually calculate the set $S$. To make the notation consistent with [11], we call the measured paths the basis paths, even if they do not necessarily form a basis in the underlying vector space as was the case in [11].

Suppose, for the moment, that the value of $\mu_{\text{max}}$ is known and equal to $D \in \mathbb{R}$. Then the following problem encodes the existence of individual edge weights ($w_e$) such that the cumulative sum ($\sum_{e \in x} w_e$) along each measured path is consistent with its measured length; that is, the measured value differs by at most $D$ from the cumulative sum.

\[\text{Problem 1. Input: DAG } G, \text{ a set of measurements } M \text{ and } D \in \mathbb{R} \]
\[
\max \quad \text{len(path)} \\
\text{s.t. } l_i - D \leq \sum_{e \in x} w_e \leq l_i + D \\
\text{for each measurement } (x_i, l_i) \in M \\
\forall e \in E \quad b_e = 1 \\
\forall e \in E \quad p_e \geq 0 \\
\forall e \in E \quad b_e \in \{0, 1\} \\
\forall e \in E \quad p_e \geq 0 \\
\]

Where $\max(\text{len(path)})$ expresses the length of the longest cumulative sum along some source-to-sink path in the graph. We now turn this problem into an ILP by expressing the existence of a path as follows:

\[\text{Problem 2. Input: DAG } G, \text{ a set of measurements } M \text{ and } D \in \mathbb{R} \]
\[
\max \quad \sum_{e \in E} p_e \\
\text{s.t. } l_i - D \leq \sum_{e \in x} w_e \leq l_i + D \\
\text{for each measurement } (x_i, l_i) \in M \\
\sum_{e \in \text{out}(s)} b_e = 1 \\
\sum_{e \in \text{in}(t)} b_e = 1 \\
\sum_{e \in (v)} b_e = \sum_{e \in \text{out}(b)} b_e \\
\forall e \in E \quad p_e \leq w_e \\
\forall e \in E \quad p_e \leq M \cdot b_e \\
\forall e \in E \quad w_e \geq 0 \\
\forall e \in E \quad b_e \in \{0, 1\} \\
\forall e \in E \quad p_e \geq 0 \\
\]

Where $M \in \mathbb{R}$ is a constant larger than any potential $w_e$. In the implementation, we take $M$ to be the largest $l_i$ in the set of measurements $S$ plus one.

In the above ILP (Problem 2), Boolean variables $b_e$’s specify which edges of the graph are present in the extremal source-to-sink path and $p_e$’s shall equal $b_e \cdot w_e$. Thus, $\sum_{e \in E} p_e$ denotes the length of the extremal source-to-sink path.

The existence of a path is encoded by the constraints specifying that there is a flow from the source to the sink. That is, that exactly one edge from the source has $b_e = 1$, exactly one edge to the sink has $b_e = 1$ and that for all intermediate vertices, the number of incoming edges to that vertex with $b_e = 1$ equals the number of outgoing edges from that vertex with $b_e = 1$.

Further, for each edge $e \in E$, we use the variable $p_e$ to denote the multiplication $p_e = b_e \cdot w_e$. As $b_e \in \{0, 1\}$, we have $p_e \leq w_e$. Also, the constraints $p_e \leq M \cdot b_e$ ensure that if $b_e = 0$ then $p_e = 0$. On the other hand, if $b_e = 1$ then constraints imply only that $0 \leq p_e \leq w_e$. Finally, note that the objective function is to maximize $\sum_{e \in E} p_e$; hence, if $b_e = 1$ then optimum value for $p_e$ is to set $p_e$ to $w_e$. Hence, in the optimal solution, if $b_e = 1$ then $p_e = w_e = 1 \cdot w_e = b_e \cdot w_e$ as desired.

Recall, that for measurement $(x_i, l_i)$ it holds that $l_i = \sum_{e \in x_i} w_e + d$ where $|d| \leq \mu_x \leq \mu_{\text{max}}$. Thus, in general,
by perturbing the measurements.

| Benchmark     | Perturbation | Perturbation |
|---------------|--------------|--------------|
| altitude      | 57.0         | 66.1         | 87.8         | 126.5        |
| stabilisation | 343.2        | 374.3        | 807.3        | 1107.2       |
| automotive    | 1116.0       | 1281.3       | 1496.5       | 2981.5       |
| cctask        | 73.9         | 110.6        | 150.9        | 270.0        |
| irobot        | 37.2         | 95.9         | 288.8        | 552.8        |
| sm            | 0.1          | 23.2         | 117.4        | 216.8        |

C. Optimality

The solution of Problem 2 is the best estimate of the longest path that is consistent with measurements \( M \). We now show how good the estimate is.

Consider the solution of Problem 2 with \( D \) equal to the solution of Problem 3. For each edge \( e \), let \( p(w_e) \) denote the value of the variable \( w_e \) in the solution, and let \( \tau \) be the path corresponding to the solution of Problem 2. Denote the length of \( \tau \) in the solution by \( p(\text{len}(\tau)) \). We now show how much \( p(\text{len}(\tau)) \) differs from the actual length of \( \tau \). Specifically, we shall show that the goodness of the estimate of the length of \( \tau \) is related to the following ILP.

| Benchmark     | Perturbation | Perturbation |
|---------------|--------------|--------------|
| altitude      | 57.0         | 66.1         | 87.8         | 126.5        |
| stabilisation | 343.2        | 374.3        | 807.3        | 1107.2       |
| automotive    | 1116.0       | 1281.3       | 1496.5       | 2981.5       |
| cctask        | 73.9         | 110.6        | 150.9        | 270.0        |
| irobot        | 37.2         | 95.9         | 288.8        | 552.8        |
| sm            | 0.1          | 23.2         | 117.4        | 216.8        |

3Note that Problem 2 is a linear program and not an integer linear program.
Note that, with this assignment to \( w_e \)'s, the length of \( \pi \) equals \( \text{len}(\pi) = 1 + \frac{1}{|E|} > 1 \). Now, consider any other path \( \tau \) used in measurements \( M \). In particular, \( \tau \neq \pi \). There are \( |E| \) edges in \( G \) and the weight \( w_e \) associated with each edge \( e \) is at least \( \frac{1}{|E|} \). Hence, \( \text{len}(\tau) \geq |E| \times -\frac{1}{|E|} = -1 \).

Now, if \( \tau \) does not include \( e_i \), that is \( e_i \notin \tau \) then, \( \text{len}(\tau) \leq 0 \) as \( w_e \) is the only positive weight. If \( \tau \) includes \( e_i \), that is \( e_i \in \tau \), then \( \tau \) necessarily contains at least one edge from \( D \) as \( \tau \) is different from \( \pi \). Hence, \( \text{len}(\tau) \leq 1 \). In any case, \(-1 \leq \text{len}(\tau) \leq 1 \) as required and thus we have given a valid assignment to \( w_e \)'s with \( \text{len}(\pi) > 1 \).

Recall that the set \( S \) denotes the set of paths occurring in measurements \( M \). Let \( r(w_e) \) and \( r(\mu_x) \) be the real values of \( w_e \) for each edge \( e \) and \( \mu_x \) for each path \( x \in S \). Then for each edge \( e \) the expression \( |p(w_e) - r(w_e)| \) denotes the difference between the calculated values of \( w_e \) and the actual value of \( w_e \). Analogously, the expression extends to entire paths: for a path \( x \) we have \( p(w_e) = \sum_{e \in x} p(w_e) \). Now, the difference for the worst path can be bounded as follows.

**Theorem 4.** Let \( k \) be the solution of Problem \( 4 \). Then

\[
|p(\text{len}(\tau)) - r(\text{len}(\tau))| \leq 2k\mu_{\text{max}}
\]

**Proof:** Note that by construction, \( r(w_e) \) and \( r(\mu_x) \) are a solution of Problem \( 3 \). Hence, for every \((x_i, l_i) \in M \) it holds that:

\[
l_i - \mu_{\text{max}} \leq l_i - r(\mu_x) \leq \sum_{e \in x_i} r(w_e) \leq l_i + r(\mu_x) \leq l_i + \mu_{\text{max}}.
\]

Recall that \( D \leq \mu_{\text{max}} \) and that \( p(w_e) \) is a solution of Problem \( 2 \). Hence, for every \((x_i, l_i) \in M \) it holds that:

\[
l_i - \mu_{\text{max}} \leq l_i - D \leq \sum_{e \in x_i} p(w_e) \leq l_i + D \leq l_i + \mu_{\text{max}}.
\]

Hence, by subtracting the last two equations from each other, we have for any basis path \( x_i \in S \) that:

\[
-2\mu_{\text{max}} \leq \sum_{e \in x_i} p(w_e) - r(w_e) \leq 2\mu_{\text{max}}.
\]

Now, dividing by \( 2\mu_{\text{max}} \) we have:

\[
-1 \leq \frac{\sum_{e \in x_i} p(w_e) - r(w_e)}{2\mu_{\text{max}}} \leq 1.
\]

for any basis path \( x_i \in S \).

Thus, the above inequality implies that taking

\[
w_e = \frac{r(w_e) - p(w_e)}{2\mu_{\text{max}}}
\]

is a (not necessarily optimal) solution of Problem \( 3 \). Since \( k \) is the length of the longest path achievable in Problem \( 3 \), it follows that for any path \( x \) (not necessarily in the basis), we have

\[
-k \leq \frac{\sum_{e \in x} p(w_e) - r(w_e)}{2\mu_{\text{max}}} \leq k.
\]

By rearranging, we have

\[
-2k\mu_{\text{max}} \leq \sum_{e \in x} p(w_e) - r(w_e) \leq 2k\mu_{\text{max}}.
\]

In other words, the calculated length differs from the real length by at most \( 2k\mu_{\text{max}} \), as desired.

Recall that the algorithm in \( 4 \) has difference between the estimated and the actual length at most \( 2|E|\mu_{\text{max}} \) whereas our algorithm has \( 2k\mu_{\text{max}} \) where \( k \) is the solution of Problem \( 4 \). Observe that the dependence in the error term on \( \mu_{\text{max}} \) is unavoidable as \( \mu_{\text{max}} \) is inherent to the timing properties of the underlying platform.

For comparison, we have generated the same basis as in \( 4 \) and calculated the corresponding \( k \)'s for several benchmarks. Table IV summarizes the results (see Table V for the description of benchmarks). As can be seen from the table, when using the same set of measurements, our method gives more accurate estimates than the one in \( 4 \).

Furthermore, recall that in Problem \( 3 \) we calculate the best (lower) bound \( D \) on \( \mu_{\text{max}} \) consistent with the given measurements. Together with the above theorem, this gives “error bounds” to the estimate in Problem \( 2 \). Specifically, if the length of the longest path computed in Problem \( 2 \) is \( T \) then, the length of the path when measured, that is consistent with the measurements is within \( T \pm (2k \times D) \). However, note that this is only the best bound deducible from the measurements since \( D \leq \mu_{\text{max}} \). Since \( \mu_{\text{max}} \) is not directly observable and we assume no nontrivial bound on \( \mu_{\text{max}} \), the length of the path cannot be bounded more accurately without actually measuring the path.

The above analysis applies to the extraction of the single longest path. Now, suppose that instead of extracting just one longest path, we want to extract \( K \) longest paths. To that end, we iterate the above procedure and whenever a path is extracted, we add a constraint eliminating the path from the solution of Problem \( 2 \) and then solve the updated ILP. For a path \( x \), the constraint eliminating it from the solution space of Problem \( 2 \) is \( \sum_{e \in x} b_e < |x| \). The constraint specifies that not all the edges along \( x \) can be taken together in the solution.

As the length of the predicted and the measured length differ, it may happen (e.g., Table V) that when measured the length of the path predicted to be the longest is not actually the longest. Thus, to find the longest path, we may

---

**TABLE II:** Comparison of the accuracy in the longest path extraction between our algorithm and the one in \( 4 \). For our accuracy, we take \( 2 \times k \) where \( k \) is the solution of Problem \( 4 \).

| Benchmark | # Basis Paths | \( \# \) L Paths | Our Accuracy |
|-----------|---------------|----------------|-------------|
| altitude  | 6             | 12             | 10.0        |
| stabilisation | 10           | 20             | 16.4        |
| automotive | 13            | 26             | 14.0        |
| cctask    | 18            | 36             | 34.0        |
| irobot    | 21            | 42             | 18.0        |
| sin       | 69            | 138            | 48.0        |

By rearranging, we have

\[
-2k\mu_{\text{max}} \leq \sum_{e \in x} p(w_e) - r(w_e) \leq 2k\mu_{\text{max}}.
\]

In other words, the calculated length differs from the real length by at most \( 2k\mu_{\text{max}} \), as desired.
need to iterate the above process by generating paths with ever smaller predicted lengths, stopping whenever the current estimate differs by more than \((2k \times D)\) from the longest estimate.

D. Basis Computation

The algorithm (Problem 2) to estimate the longest path depends on the set of measurements \(M\) of the basis paths \(S\). In this section we show how to calculate such a set of paths. In general, arbitrary sets of paths can be used as basis paths. For example, we have shown in Table III that using the set of paths used in [11], we are able to get more accurate estimates than those obtained in [11].

Recall that (Theorem 4) the accuracy of the solution of Problem 2 is tightly coupled with the solution of Problem 4. This leads to a tunable algorithm, which depending on the desired accuracy of the predictions, calculates a set of paths to be used in Problem 2.

Specifically, given a desired accuracy \(A \in \mathbb{R}, A \geq 1\), we want to find a set of feasible paths \(S\) such that the solution of Problem 4 is at most \(A\). We implemented a simple iterative algorithm (Algorithm 1) that finds such a set by repeatedly extending the set of paths by the path corresponding to the solution of Problem 4.

In the algorithm, if the longest extracted path is infeasible in the underlying program, we add a constraint into the ILP (Problem 4) prohibiting the path. That is, if the longest path is infeasible and \(\tau\) is the unsatisifiable core of the longest path\(^7\), then we add a constraint that not all the \(b_e\),s corresponding to the edges used in \(\tau\) are set to 1 in Problem 2, i.e., \(\sum_{e \in \tau} b_e < |\tau|\) where \(|\tau|\) denotes the number of edges in \(\tau\). Then we solve the updated ILP.

\[
S \leftarrow \emptyset \\
\text{while} \ (\text{Solution of Problem 4 with paths } S) > A \text{ do} \\
\quad x \leftarrow \text{longest path in Problem 4} \\
\quad \text{if } x \text{ is feasible then} \\
\quad \quad S \leftarrow S \cup \{x\} \\
\quad \text{else} \\
\quad \quad \text{Add a constraint prohibiting } x \\
\quad \text{end} \\
\text{return } S
\]

Algorithm 1: Iterative algorithm for basis computation

**Theorem 5.** If \(A \geq 1\) then the Algorithm 1 terminates with a set \(P\) of paths such that the solution of Problem 4 with paths \(P\) is at most \(A\).

**Proof:** Note that each constraint in Problem 4 limits the length of some path to (at most) 1. In particular, if \(S\) contains all the paths in the graph then the solution of Problem 4 equals 1.

\(^7\)Minimal set of edges that cannot be taken together as identified by an SMT solver.

| Benchmark | Desired \(k\) | Actual \(k\) | # Basis Paths | Time(s) |
|-----------|--------------|--------------|---------------|---------|
| altitude  | 10           | 5.0          | 7             | 0.03    |
|           | 5            | 5.0          | 7             | 0.03    |
|           | 2            | 1.0          | 10            | 1.66    |
| stabilisation | 10         | 7.0          | 11            | 0.10    |
|           | 5            | 4.7          | 12            | 0.96    |
|           | 2            | 2.0          | 40            | 22.78   |
| automotive | 10           | 7.0          | 14            | 0.14    |
|           | 5            | 5.0          | 14            | 0.89    |
|           | 2            | 2.0          | 30            | 27.60   |
| cctask    | 10           | 9.0          | 19            | 0.20    |
|           | 5            | 5.0          | 25            | 4.10    |
|           | 2            | 2.0          | 76            | 42.91   |
| irobot    | 10           | 9.0          | 22            | 0.50    |
|           | 5            | 5.0          | 34            | 20.13   |
|           | 2            | 2.0          | 118           | 182.42  |
| sm        | 22           | 21.8         | 70            | 328.14  |
|           | 18           | 18.9         | 73            | 7089.04 |
|           | 15           | 14.5         | 77            | 10331.49|

Further, if the algorithm finds some path \(x\) to be the longest in some iteration then the length of \(x\) in all subsequent iterations will be at most 1 as \(x \in S\). Therefore, as long as the solution is greater than 1, the longest path found is different from all the paths found in the previous iterations.

Also, if the path is infeasible, then a constraint is added that prevents the path occurring in any subsequent iterations. It follows from these considerations that the algorithm keeps adding new paths in each iteration and eventually terminates.

By construction, the solution of Problem 4 with the set of paths \(S\) is at most \(A\).

In the extreme case of \(A = 1\), it immediately follows from Theorem 5 that Algorithm 1 returns all feasible paths in the underlying graph.

We have implemented the above iterative algorithm and evaluated it on several case studies. Table III summarizes the number of paths generated by the algorithm for the given accuracy \(k\) as well as the running time required to find those paths.

We have observed that the basis computation took substantial part of the entire algorithm. However, notice that the basis-computation algorithm (Algorithm 1) need not start with \(S = \emptyset\) and works correctly for any initial collection of paths \(S\). Therefore, as an optimization, we first compute the initial set of paths \(S\) using the original algorithm from [11], which computes the 2-barycentric spanner of the underlying DAG. Only then we proceed with the iterative algorithm to find a set of paths with the desired accuracy.

Figure 4 shows the performance of the iterative algorithm on two benchmarks. The decreasing (blue) line shows how the accuracy \(k\) decreases with each path added to the basis. The increasing (red) line shows the time needed to find each path. The figure shows only the performance after the precomputation of the 2-barycentric spanner.
The decreasing (blue) line shows the length of $x$ computed ($k$) in line 3. The increasing (red) line shows the time taken to perform each iteration.

(a) cctask

(b) irobot

IV. Evaluation

A. Implementation

The algorithm to identify the longest (up to accuracy $k$) path a given program $P$ is shown in Algorithm 2. We now briefly describe the implementation of main stages of the Algorithm 2. Our implementation is build on top of [11]. See [11] for further discussion of technical details.

CFG extraction The procedure begins by building the CFG $G$ of the given program. The vertices correspond to locations and edges to individual statements. The extraction is build on top of CIL front end for C [14]. Note that all ILPs (Problems 2, 3, and 4) introduce a variable per each edge of $G$ and that each problem optimizes for the length of a source-to-sink path. Thus, if some edge is always followed by another one (the in- and out-degree of the joining vertex is one) then the edges can be merged into a single one without changing the solution of the ILP problems. Therefore, we process $G$ by merging edges that lie on a single path into a single edge. This reduces the number of variables used and improves the performance.

Basis computation The basis is computed as described in Section III-D, Algorithm 1. We use Gurobi solver [15] to solve individual ILPs and use the 2-barycentric spanner as the initial set $S$. Solving the ILPs posed the main bottleneck of our approach.

Input generation Each path through the CFG $G$ corresponds to a sequence of operations in the program and hence a conjunction of statements. For a given path, we use the Z3 SMT solver [16] to find an input that corresponds to the given path or to prove that the path is infeasible and no corresponding input exists. In the experiments, SMT solving was fairly fast.

Longest-path extraction We solve Problem 2 using Gurobi ILP solver [15]. If the extracted path $\pi$ is infeasible, we add a constraint ($\sum_{e \in \pi} b_e < |\pi|$) eliminating the path from the solution and solve the new problem. Similarly, we solve for $K$ longest paths; we add a constraint prohibiting the extracted path and solve the resulting ILP again, repeating until we successfully generate $K$ feasible paths.

Recall that the calculated length of the path is accurate only up to the precision $2k \times D$. Thus, we can repeat the process until the length of the extracted path is outside of the range for the longest extracted path. However, in practice we did not find this necessary and found the longest path after a few iterations.

Path-Length Measurement To measure the execution time of the given program on a given input, we create a C program where we set the input to the given values and then run it using a cycle-accurate simulator of the PTARM processor [17].

B. Benchmarks

We have evaluated our algorithm on several benchmarks and compared it with the algorithm in [11]. We used the same benchmarks as in [11] as well as benchmarks from control tasks from robotic and automotive settings. The benchmarks in [11] come from Mälardalen benchmark suite [18] and the

For example, in Figure 2, every diamond can be replaced by two edges, one edge for the top half and one edge for the bottom half.
TABLE IV: Number of nodes, edges and paths in the CFGs extracted from the benchmarks

| Benchmark | # Nodes | # Edges | # Paths |
|-----------|---------|---------|---------|
| altitude  | 36      | 40      | 11      |
| stabilisation | 64 | 72      | 216     |
| automotive | 88      | 100     | 506     |
| cctask    | 102     | 118     | 657     |
| inobot    | 170     | 195     | 8136    |
| sm        | 452     | 523     | 33,755,520 |

PapaBench suite [19]. The authors of [11] chose implementations of actual realtime systems (as opposed to hand-crafted ones) that have several paths, were of various sizes, but do not require automatic estimation of loop bounds.

Since we assume all programs contain only bounded loops and no recursion, we preprocessed the programs by unrolling the loops and inlining the functions where necessary. Table IV summarizes properties of the benchmarks used.

Table VII shows the lengths of five longest paths as found by our algorithm and the one in [11]. The first half of the table shows the results as obtained by the algorithm in [11]. The second half shows the results as obtained (together with the “error bounds”) by our algorithm (Algorithm 2).

Note that in each case the longest path returned by our algorithm is never shorter than the longest path found by [11]. In half of the cases, our algorithm is able to find a path longer than the one in [11]. Also notice that the actual measured length is always within the computed “error bounds”.

The biggest benchmark, sm, is a collection of nested switch-case logic setting state variables but with minimal computations otherwise. Hence, there is a large number of paths (33,755,520) in the CFG yet, as expected, the computed $D$ is small.

V. CONCLUSION

In this paper, we have addressed the problem of estimating the worst-case timing of a program via systematic testing on the target platform. Our approach not only generates an estimate of worst-case timing, but can also produces test cases showing how that timing is exhibited on the platform. Our approach improves the accuracy of the previously published GameTime approach, while also providing error bounds on the estimate.

Note that our approach can be adapted to produce timing estimates along arbitrary program paths. In order to do this, one can fix variables $b_i$ in Problem 2 suitably. Thus, we can also estimate the longest execution of a given path that is consistent with the measurements.

In the paper we have analyzed the timing behavior of a given program. However, instead of measuring cycles we can measure energy consumption of the program executions. The same techniques can then be applied to find the input consuming the most energy. In general, the approach presented in this paper can also be extended to other quantitative properties of the program and is not limited only to the WCET analysis.

REFERENCES

[1] Y.-T. S. Li and S. Malik, "Performance Analysis of Real-Time Embedded Software." Kluwer Academic, 1999.
[2] Reinhard Wilhelm et al., "The Determination of Worst-Case Execution Times—Overview of the Methods and Survey of Tools," ACM Transactions on Embedded Computing Systems (TECS), 2007.
[3] E. A. Lee and S. A. Seshia, "Introduction to Embedded Systems: A Cyber-Physical Systems Approach," first edition ed. http://leeseshia.org, 2011.
[4] E. A. Lee, "Computing foundations and practice for cyber-physical systems: A preliminary report," University of California at Berkeley, Tech. Rep. UCB/EECS-2007-72, May 2007.
[5] R. Kimner and P. Puschner, "Obstacles in worst-case execution time analysis," in ISORC, 2008, pp. 333–339.
[6] NASA Engineering and Safety Center. "NASA report on Toyota unintended acceleration investigation, appendix a: Software," [http://www.nhtsa.gov/staticfiles/nvs/pdf/NASA_FR_Appendix_A_Software.pdf].
[7] S. A. Seshia and A. Rakhlin, "Game-theoretic timing analysis," in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2008, pp. 575–582.
[8] ——, "Quantitative analysis of systems using game-theoretic learning," ACM Transactions on Embedded Computing Systems (TECS), 2012.
[9] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in Proceedings of the 32Nd Annual ACM/IEEE Design Automation Conference, ser. DAC ’95. New York, NY, USA: ACM, 1995, pp. 456–461. [Online]. Available: http://doi.acm.org/10.1145/217474.217570.
[10] R. Wilhelm, "Determining bounds on execution times." 2009.
[11] S. A. Seshia and A. Rakhlin, "Game-theoretic timing analysis," in Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, ser. ICCAD ’08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 575–582. [Online]. Available: http://dl.acm.org/citation.cfm?id=1509456.1509584.
[12] Z. Wasson, "Analyzing data-dependent timing and timing repeatability with gametime," Master’s thesis, EECS Department, University of California, Berkeley, May 2014. [Online]. Available: [http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-132.html]
[13] B. Awerbuch and R. D. Kleinberg. "Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches," in Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing, ser. STOC ’04. New York, NY, USA: ACM, 2004, pp. 45–53. [Online]. Available: http://doi.acm.org/10.1145/1007352.1007367.
[14] George Necula et al., "CIL - infrastructure for C program analysis and transformation," [http://manju.cs.berkeley.edu/cil/].
[15] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: [http://www.gurobi.com].
[16] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, ser. TACAS ’08/ETAPS ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. [Online]. Available: [http://dl.acm.org/citation.cfm?id=1792274.1792266].
[17] Center for Hybrid and Embedded Software, UC Berkeley, "The PTARM simulator," [http://chess.eecs.berkeley.edu/pret/].
[18] Mälardalen WCET Research Group, "The Mälardalen benchmark suite," [http://www.mrtc.mdh.se/projects/wcet/benchmarks.html].
[19] F. Nemer, H. Cass, P. Sainrat, J. paul Bahsoun, and M. D. Michiel, "Papabench: a free real-time benchmark," in In WCET ’06, 2006.
TABLE V: Comparison of the results produced by the algorithm presented in this paper and in [11] for generating the top five longest paths for a set of benchmarks. Column Predicted shows the length predicted by each algorithm. Column Measured show the running time measured when run on the corresponding input. For our paper, we give “error bounds” of the form $k \times D$. The column Time shows the time it takes to generate the estimates and test cases. The largest measured value per each benchmark is shown in bold.

| Benchmark | GameTime [11] | Our Algorithm |
|-----------|---------------|---------------|
|           | Predicted     | Measured      | Predicted     | Measured      | Time(s) | Predicted     | Measured      | Time(s) |
| altitude  | 867           | 867           | 909 ± 1.0 × 57.0 | 867           | 11.7     | 815 ± 1.0 × 57.0 | 768           | 14.4 |
|           | 789           | 789           | 729 ± 1.0 × 57.0 | 789           | 743      | 719 ± 1.0 × 57.0 | 751           | 77.2 |
|           | 659           | 763           | 598 ± 1.0 × 57.0 | 581           | 867      | 638 ± 1.0 × 57.0 | 581           | 867 |
|           | 4387          | 3903          | 4903 ± 2.0 × 343.0 | 3999          | 26.4     | 4020 ± 2.0 × 343.0 | 4046          | 3999 |
|           | 4293          | 4036          | 4285 ± 2.0 × 343.0 | 3944          | 4290     | 4284 ± 2.0 × 343.0 | 3516          | 4284 |
|           | 4286          | 3212          | 4284 ± 2.0 × 343.0 | 3516          | 4196     | 4248 ± 2.0 × 343.0 | 3697          | 4248 |
| stabilisation | 11515         | 11515         | 11515 ± 2.0 × 1116.0 | 11515         | 47.0     | 11515 ± 2.0 × 1116.0 | 11515         | 93.8 |
|           | 13595         | 8106          | 11824 ± 2.0 × 1116.0 | 10982         | 11614    | 11696 ± 2.0 × 1116.0 | 10657         | 11614 |
|           | 11515         | 11515         | 11624 ± 2.0 × 1116.0 | 10577         | 11515    | 11515 ± 2.0 × 1116.0 | 11515         |
|           | 11243         | 11138         | 9830 ± 2.0 × 1116.0 | 9263          |          |                      |               |      |
| automotive | 991           | 808           | 870 ± 2.0 × 73.0  | 861           | 29.4     | 869 ± 2.0 × 73.0  | 858           | 138.4 |
|           | 972           | 605           | 866 ± 2.0 × 73.0  | 897           | 943      | 865 ± 2.0 × 73.0  | 897           | 943   |
|           | 936           | 848           | 861 ± 2.0 × 73.0  | 873           | 924      | 861 ± 2.0 × 73.0  | 873           | 924   |
| cctask    | 1462          | 1430          | 1451 ± 2.0 × 37.0 | 1406          | 1459     | 1459 ± 2.0 × 37.0 | 1411          |      |
|           | 1457          | 1418          | 1449 ± 2.0 × 37.0 | 1411          | 1454     | 1448 ± 2.0 × 37.0 | 1411          |
|           | 1454          | 1463          | 1448 ± 2.0 × 37.0 | 1464          | 1454     | 1448 ± 2.0 × 37.0 | 1411          |
| irobot    | 2553          | 2550          | 2552 ± 21.81 x 0.2 | 2550          | 50.9     | 2552 ± 21.81 x 0.2 | 2550          | 269.0 |
|           | 2554          | 2550          | 2551 ± 21.81 x 0.2 | 2550          |          |                      |               |      |
|           | 2536          | 2537          | 2536 ± 21.81 x 0.2 | 2537          |          |                      |               |      |
|           | 2534          | 2537          | 2536 ± 21.81 x 0.2 | 2537          |
|           | 2532          | 2537          | 2531 ± 21.81 x 0.2 | 2537          |
| sm        | 2553          | 2550          | 2552 ± 21.81 x 0.2 | 2550          | 211.0    | 2551 ± 21.81 x 0.2 | 2550          | 3290.2 |
|           | 2554          | 2550          | 2551 ± 21.81 x 0.2 | 2550          |||