Game-Theoretic Timing Analysis

Sanjit A. Seshia and Alexander Rakhlin
EECS Department, UC Berkeley
{sseshia,rakhlin}@eeecs.berkeley.edu

Abstract—Estimating the worst-case execution time (WCET) of tasks is a key step in the design of reliable real-time software and systems. In this paper, we present a new, game-theoretic approach to estimating WCET based on performing directed measurements on the target platform. We model the estimation problem as a game between our algorithm (player) and the environment of the program (adversary), where the player seeks to find the longest path through the program while the adversary sets environment parameters to thwart the player. We present both theoretical and experimental results demonstrating the utility of our approach. On the theoretical side, we prove that our algorithm can converge to find the longest path with high probability. Experimental results indicate that our approach is competitive with an existing technique based on static analysis and integer programming. Moreover, the approach can be easily applied to even complex hardware/software platforms.

I. INTRODUCTION

Timing analysis plays a central role in the design of real-time embedded systems. The main problem is to determine the worst-case execution time (WCET) of programs, commonly referred to as tasks, and then use the result to provide timing guarantees on the system composed of these tasks. WCET estimates are core components of any methodology to design reliable real-time systems, since they are used in scheduling algorithms and to give formal guarantees on real-time performance and correctness (see, e.g., [1]).

There has been much research on the WCET estimation problem over the last 20 years [2], [3]. The overall problem has two dimensions: the path problem, which is to find the worst-case path through the task, and the state problem, which seeks to find the worst-case environment state to run the task from. Significant progress has been made, especially in the computation of bounds on loops in tasks, in modeling the dependencies amongst program fragments using (linear) constraints, and modeling some aspects of processor behavior. However, as pointed out in recent papers by Lee [4] and Kirner and Puschner [5], it is becoming increasingly difficult to precisely model the complexities of the underlying hardware platform (e.g., out-of-order processors with deep pipelines, branch prediction, caches, parallelism) as well as the software environment. This results in WCET bounds that are either too pessimistic or too optimistic. Additionally, the evaluation of WCET estimation tools is ad-hoc and based on performing random, unguided measurements and seeing how close to the bound these measurements get. As Kirner and Puschner [5] write, a major challenge for measurement-based techniques is the automatic and systematic generation of test data.

We address these challenges by presenting GAME TIME, a measurement-based technique for WCET analysis based on a novel game-theoretic paradigm. We model the estimation problem as a game between our WCET estimation algorithm (player) and the environment of the program (adversary), where the player seeks to find the longest path through the program while the adversary sets environment parameters to thwart the player. Over many rounds of the game, our algorithm learns enough about the environment to be able to predict the longest path with high probability. Our algorithm is not only robust to adversarial choices made by the environment, but also to errors in measurement.

A key idea in our approach is to perform directed measurements of the program, by sampling only so-called basis paths. The idea is that the length of any program path can be estimated as a linear combination of the observed lengths of the basis paths. We believe that this concept of basis paths and our estimation algorithm are useful concepts for quantitative analysis in general, both for software as well as for circuits.

We present both theoretical and experimental results demonstrating the utility of our approach. On the theoretical side, we prove that if we run our algorithm “long enough” (formalized in Section IV), it can find the longest path in the task with high probability. Once the longest path has been found, it can be repeatedly run to estimate the worst-case execution time. Our results enable us to find not just a single longest path, but also paths of length within $\epsilon$ of the longest.

We have implemented our approach in a tool called GAME TIME. We present experimental results comparing GAME TIME to the WCET estimation tool Chronos [6], and results indicate that our approach is competitive with existing techniques based on static analysis and integer programming, without incurring the difficulties involved in modeling complex processor behavior. Moreover, since the approach is measurement-based, it is easy to apply to varied and complex platforms.

The outline of the paper is as follows. We begin with a survey of related work in Section II. The basic formulation and an overview of our approach is given in Section III. The algorithm and main theorems are given in Section IV, and experimental results in Section V. We conclude with a discussion of other applications of this work in Section VI.

II. BACKGROUND AND RELATED WORK

We briefly review literature on WCET estimation and results from learning theory that our algorithms are based upon.

A. WCET Estimation

There is a vast literature on WCET estimation, comprehensively surveyed by Li and Malik [2] and Wilhelm et al. [3], [7]. For lack of space, we only include here a brief discussion of current approaches and do not cover all tools. References to current techniques can be found in a recent survey [3].

There are two parts to current WCET estimation methods: program path analysis (also called control flow analysis) and processor behavior analysis. In program path analysis, the tool tries to find the program path that exhibits worst-case execution time. In processor behavior analysis (PBA), one models the details of the platform that the program will execute on, so as to be able to predict environment behavior such as cache misses and branch mis-predictions that determine
comes from the realization that well-performing algorithms can be found (a) for large decision spaces, such as paths in basic "bandit" problem was put forth by Robbins [9] in 1952 and has been well-understood since then. The recent progress and Lugosi [10] for a comprehensive treatment of sequential prediction. Some relevant results can be found in [11]–[13].

In comparison, our technique is measurement-based, and hence suffers no over-estimation. It is distinct from existing measurement-based techniques due to the novel game-theoretic formulation and use of current results from learning theory. Our approach does rely on some static techniques, in deriving loop bounds and using symbolic execution and satisfiability solvers to compute inputs to drive the program down a specific path of interest. In particular, note that our approach completely avoids the difficulties of processor behavior analysis, instead directly executing the program on its target platform.

While adversarial analysis has been employed for related problems, such as system-level dynamic power management [8], to our knowledge, the adversarial analysis technique in this paper is the first for timing estimation and for estimating quantitative parameters of programs.

B. Learning Theory

Results of this paper build on the game-theoretic linear prediction literature in learning theory. This field has witnessed an increasing interest in sequential (or online) learning, whereby an agent discovers the world by repeatedly acting and receiving feedback. Of particular interest is the problem of learning in the presence of an adversary with a complete absence of statistical assumptions on the nature of the observed data.

The problem of sequentially choosing paths to minimize the regret (the difference between cumulative lengths of the paths chosen by our algorithm and the total length of the longest path after \( T \) rounds) is known as an instance of bandit online linear optimization. The “bandit” part of the name is due to the connection with the multi-armed bandit problem, where only the payoff of the chosen “arm” (path) is revealed. The basic “bandit” problem was put forth by Robbins [9] in 1952 and has been well-understood since then. The recent progress comes from the realization that well-performing algorithms can be found (a) for large decision spaces, such as paths in a graph, and (b) under adversarial conditions rather than the stochastic formulation of Robbins. We believe it is useful to bring out these results for the problem of timing analysis.

We refer the reader to the recent book of Cesa-Bianchi and Lugosi [10] for a comprehensive treatment of sequential prediction. Some relevant results can be found in [11]–[13].

III. THEORETICAL FORMULATION AND OVERVIEW

The worst-case execution time (WCET) estimation problem can be defined as follows:

Given a terminating software task \( S \) and a platform \( M \) on which \( S \) executes, estimate the longest time \( S \) takes to terminate on \( M \).

A central idea in our theoretical formulation is that the platform can be treated as an adversary about which we learn over time through repeated experimentation. The learned information is then used to predict the program path that corresponds to the worst-case execution time. This game-theoretic approach is in contrast to the traditional approach of modeling a-priori all the complexities of the platform.

The main ideas in our theoretical formulation are elaborated below.

Game-theoretic formulation: We model the WCET estimation problem as a game between the WCET estimation tool \( T \) and the environment \( E \) of \( S \).

The game proceeds over multiple rounds, \( t = 1, 2, 3, \ldots \). In each round, \( T \) picks the inputs to \( S \). These inputs determine the path taken through the program. Simultaneously, \( E \) adversarially picks environment parameters, such as the state of the cache before running \( S \). This choice by \( E \) can depend on the inputs selected by \( T \).

At the end of each round \( t \), \( T \) receives as feedback the execution time \( l_t \) of \( S \) for its chosen path under the parameters chosen by \( E \). Note that we assume that \( T \) only receives the overall execution time of the task, not a more fine-grained measurement of (say) each basic block in the task along the chosen path. This enables us to minimize any skew from instrumentation inserted to measure time.

Based on this feedback \( l_t \), \( T \) must modify its input-selection strategy to improve its chances of picking the inputs that trigger the WCET of the task.

The goal of \( T \) is to select inputs so that within a time horizon \( T \) it can accumulate enough data to identify, with high probability, the longest execution time of \( S \) that could have been exhibited during rounds \( t = 1, 2, \ldots, T \).

Note that this longest execution time need not be due to inputs tried out by \( T \).

By permitting \( E \) to select environment parameters based on \( T \)’s choice of path, we can model path-dependent timing as well as perturbation in execution time of a single path due to variation in environmental conditions or measurement error. The more predictable the timing behavior of the platform, the smaller this perturbation will be. For theoretical analysis, we model the perturbation as a random variable whose mean is bounded by a parameter \( \mu_{\text{max}} \). If a platform has predictable timing, such as the PRET processor proposed by Edwards and Lee [14], it would mean that \( \mu_{\text{max}} \) is small. (The \( \mu_{\text{max}} \) parameter will play a role in determining the rate of convergence of our proposed algorithm.)

Formulation as a graph problem: An additional aspect of our model is that the game operates on the control-flow graph \( G_S \) of the task \( S \) (with loops unrolled).

In this setting, the game described above works as follows. At any round \( t \), the player \( T \) selects a path \( x_t \) through the graph \( G_S \) from a designated source node (entry point of the function) to a designated sink node (exit point/return statement of the function). This is performed by picking input values for \( S \) that drive execution down path \( x_t \). \( E \) selects lengths for all source-sink paths in \( G_S \), where this selection can depend on the choice of \( x_t \). However, \( E \) only reveals the length \( l_t \) of the chosen path \( x_t \).
The goal of $\mathcal{T}$ is thus to select paths so that within a time horizon $T$ it can accumulate enough data to identify, with high probability, the longest path in $G_s$ during rounds $t = 1, 2, \ldots, T$.

For ease of theoretical analysis, we will assume that $\mathcal{E}$ initially chooses the worst-case (longest) times for each path, and then, upon observing $x_t$, perturbs the chosen time for $x_t$ by $p_t$ to obtain $\ell_t$. While this assumption does not lose any generality in modeling the WCET problem, it allows us to cleanly factor out the term $p_t$ that represents the timing predictability of the platform.

We next give an overview of our approach.

**Overview of Our Approach**

We describe the working of our approach using a small program from an actual real-time system, the Paparazzi unmanned aerial vehicle (UAV) project [15]. Figure 1 shows the C source code for the altitude control task in the Paparazzi code, which is publicly available open source.

```c
void altitude_control_task(void) {
if (pprz_mode == PPRZ_MODE_AUTO2
  | pprz_mode == PPRZ_MODE_HOME) {
  if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
    // inlined below: function attitude_pid_run(); */
    float err = estimator_z - desired_altitude;
    desired_climb = pre_climb + altitude_pgain * err;
    if (desired_climb < -CLIMB_MAX)
      desired_climb = -CLIMB_MAX;
    if (desired_climb > CLIMB_MAX)
      desired_climb = CLIMB_MAX;
  }
}
}
```

Fig. 1. Source code for altitude control task

Starting with the source code for a task, and all the libraries and other definitions it relies on, we run the task through a C pre-processor and the CIL front-end [16] and extract the control-flow graph (CFG). In this graph, each node corresponds to the start of a basic block and edges are labeled with the basic block code or conditional statements that govern control flow. Figure 2 shows the CFG for the code shown in Figure 1.

Note that we assume that code terminates, and bounds are known on all loops. Thus, we start with code with all loops (if any) unrolled, and the CFG is thus a directed acyclic graph (DAG). We also pre-process the CFG so that it has exactly one source and one sink. Each execution through the program is a source-to-sink path in the CFG.

An exhaustive approach to program path analysis will need to enumerate all paths in this DAG. However, it is well-known that even a DAG can have exponentially many paths (in the number of vertices/edges). Thus, a brute-force enumeration of paths is not going to be efficient.

Our approach is to sample a set of basis paths. The key idea is to view each source-sink path as a vector in $\{0, 1\}^m$, where $m$ is the number of edges in the program. The $i$th entry of the vector for a path $x$ corresponds to edge $i$ of the CFG, and is 1 if edge $i$ is in $x$ and 0 otherwise. The set of all valid source-sink paths thus forms a subset $\mathcal{P}$ of $\mathbb{R}^m$.

We compute the basis for $\mathcal{P}$ in which each element of the basis is a source-sink path. Figure 3 illustrates the ideas using a simple “2-diamond” example of a CFG. In this example, paths $x_1$, $x_2$ and $x_3$ form a basis and $x_4$ can be expressed as the linear combination $x_1 + x_2 - x_3$.

Our algorithm, described in detail in Section IV, randomly samples basis paths of the CFG and drives program execution down those paths by generating tests using symbolic execution. From the observed lengths of those paths, we estimate edge weights on the entire graph. This estimate, accumulated over several rounds of the game, is then used to predict the longest source-sink path in the CFG. Theoretical guarantees on performance are proved in Section IV and experimental evidence for its utility is given in Section V.

**IV. Algorithm and Theoretical Results**

Let $\mathcal{P}$ be the set of paths between source $u$ and sink $v$ in the directed acyclic graph $G = (V, E)$. We associate each of the paths with a binary vector with $m = |E|$ components, depending on whether the edge is present or not. The path prediction interaction is modeled as a repeated game between our algorithm (Player) and the program environment (Adversary). On each round $t$, we choose a path $x_t \in \mathcal{P} \subseteq \{0, 1\}^m$ between $u$ and $v$. The adversary independently chooses the lengths of paths in the graph. We assume that this choice is made by first choosing the worst-case delays, or weights, $w_t \in \mathbb{R}^m$ on the edges of $G$, and then perturbing the overall path length exhibited to us (Player). In other words, the true worst-case length of the chosen path is $x_t^T w_t$, the dot product down those paths by generating tests using symbolic execution.
ways of getting to a statement. Indeed, a key feature of the algorithm is the ability to exploit correlations between paths to guarantee that we find the longest. Hence, we need a barycentric spanner (introduced by Awerbuch and Kleinberg [13]), a set of up to $m$ paths with two valuable properties: any path in the graph can be written as a linear combination of the paths in the spanner, and the coefficients in this linear combination are bounded in absolute value. The first requirement says that the spanner is a good representation for the exponentially-large set of possible paths; the second says that lengths of some of the paths in the spanner will be of the same order of magnitude as the length of the longest path. These properties enable us to repeatedly sample from the barycentric spanner and reconstruct delays on the whole graph. We then employ concentration inequalities\footnote{Concentration inequalities are sharp probabilistic guarantees on the deviation of a function of random variables from its mean.} to prove that these reconstructions, on average, converge to the true delays of the paths. Once we have a good statistical estimate of the true weights on all the edges, it only remains to run a longest-path algorithm (linear-time LONGEST-PATH for directed acyclic graphs).

The existence of a barycentric spanner has been shown in Awerbuch and Kleinberg [13]. In particular, the authors provide the following procedure to find a 2-barycentric spanner set (where coefficients are bounded in absolute value by 2) \{$b_1, \ldots, b_m$\} $\in \mathcal{P}$ (see also [11]).

**Algorithm 1** Finding a 2-Barycentric Spanner

1: \{$b_1, \ldots, b_m$\} $\leftarrow$ \{$e_1, \ldots, e_m$\}.
2: // Compute a basis of $\mathcal{P}$:
3: for $i = 1$ to $m$ do
4: \ $b_i \leftarrow \arg \max_{x \in \mathcal{P}} |\det(x, B_{-i})|$
5: end for
6: // Transform $B$ into a 2-barycentric spanner:
7: while $\exists x \in \mathcal{P}, i \in \{1, \ldots, m\}$ satisfying $|\det(x, B_{-i})| > 2|\det(b_i, B_{-i})|$ do
8: \ $b_i \leftarrow x$
9: end while

In Algorithm 1, $B = (b_1, \ldots, b_m)$ and $B_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_m)$. The output of the algorithm is a 2-barycentric spanner $B$; i.e., any path $x \in \mathcal{P}$ can be written as $x = \sum_{i=1}^{m} \alpha_i b_i$ with $|\alpha_i| \leq 2$. It is shown [13] that the running time of Algorithm 1 is only quadratic in $m$. Gyorgy et al. [12] extend the above procedure to the case where the set of paths spans only a $b$-dimensional subspace of $\mathbb{R}^m$ (where $b \leq m$), a scenario which is more realistic for our setting. Slightly abusing notation, let $B$ be the $b \times m$ matrix with $b_i$’s as rows. We define the Moore-Penrose pseudo-inverse of $B$ as $B^+ = B^T (BB^T)^{-1}$. It holds that $BB^+ = I_b$.\\

**B. The Algorithm**

Our algorithm, called GAME TIME, is given below as Algorithm 2. For theoretical analysis, let $M$ be any upper bound on the length of the longest path. Before specifying the algorithm, we make the following assumptions on the the additive factors $p_t$, as discussed in Section III. If a path $x_t$ is chosen, $p_t$ is a random sample such that $|\mathbb{E}[p_t|x_t]| \leq \mu_{max}$ and for simplicity assume that this distribution (conditional on $x_t$) has support...
over a bounded interval $[-N,0]$. (Note however, that we place no restrictions on the value of $N$.)

**Algorithm 2 GAME-TIME**

1. Input $\tau \in \mathbb{N}$, $\delta > 0$, $\rho > 0$
2. Compute a 2-barycentric spanner $\{b_1, \ldots, b_b\}$
3. for $t = 1$ to $\tau$ do
4. Environment chooses $w_t$ in an adversarial manner.
5. We choose $i_t \in \{1, \ldots, b\}$ uniformly at random.
6. Environment chooses a distribution with support $[-N,0]$ and mean $\mu_{i_t} \geq -\mu_{\text{max}}$, draws $p_t$.
7. We predict the path $x_t = b_{i_t}$ and observe the path length $l_t = b_{i_t}^\top w_t + p_t$.
8. Estimate $\tilde{v}_t \in \mathbb{R}^m$ as $\tilde{v}_t = b_{i_t} \cdot e_t$, where $\{e_t\}$ denotes the standard basis.
9. Compute estimated weights $\tilde{w}_t = B^\top \tilde{v}_t$
10. end for

11. Use the obtained sequence $\tilde{w}_1 \ldots \tilde{w}_\tau$ to find a longest path(s). For example, for Theorem 4.2, we compute $x_\tau^\ast := \arg \max_{x \in \mathcal{P}} x^\top \sum_{t=1}^{\tau} \tilde{w}_t$.

Since we have assumed an adaptive adversary that produces $w_t$ based on our previous choices $x_1 \ldots x_{t-1}$ as well as the random factors $p_1 \ldots p_{t-1}$, we should take care in dealing with expectations. Let us denote the conditional expectation $\mathbb{E}_n[A] = \mathbb{E}[A|i_1, \ldots, i_{t-1}, p_1, \ldots, p_{t-1}]$, keeping in mind that randomness at time $t$ stems from our random choice $i_t$ and the adversary’s random choice $p_t$ given $i_t$. We stress that the adversary can vary not only $p_t$, but also the distribution of $p_t$, according to the path chosen by the Player.

The following Lemma is key to proving that Algorithm 2 performs well. It quantifies the deviations of our estimates of the delays on the whole graph, $\tilde{w}_t$, from the true delays $w_t$ (which we cannot observe).

**Lemma 4.1:** With probability at least $1 - \delta$, for all $x \in \mathcal{P}$,

\[
\frac{1}{\tau} \sum_{t=1}^{\tau} (\tilde{w}_t - w_t)^\top x \leq 2b\mu_{\text{max}} + \tau^{-1/2}c\sqrt{2b + 2\ln(2\delta^{-1})},
\]

where $c = 2b(2M + N + \mu_{\text{max}})$.

**Proof:** We will show that $\mathbb{E}_n\tilde{w}_t x \approx w_t x$ for any $x \in \mathcal{P}$, i.e. the estimates are almost unbiased modulo the perturbation $p_t$ on the subspace spanned by $\{b_1, \ldots, b_b\}$.

Define $v_t = B w_t$ just as $\tilde{v}_t = B\tilde{w}_t$. It holds that the bias of $\tilde{v}_t$ as an estimator of $v_t$ is exactly the bias introduced by the adversary through $p_t$’s. Indeed, taking expectations with respect to $i_t$ and $p_t$,

\[
\mathbb{E}_n\tilde{v}_t = \mathbb{E}_n \mathbb{E}_{i_t} \mathbb{E}_{p_t}[b(b_i^\top w_t + p_t) \cdot e_t|i_t] = \frac{1}{b} \sum_{i=1}^{b} b(b_i^\top w_t) \cdot e_i + \sum_{i=1}^{b} \mu_i e_i = Bw_t + \mu_t = v_t + \mu_t
\]

where $\mu_t$ is the vector of biases chosen by the adversary for round $t$.

2For random variables $X$ and $\hat{X}$, $\hat{X}$ is said to be an unbiased estimate of $X$ if $\mathbb{E}[X - \hat{X}] = 0$.

Fix any $\alpha \in \{-2,2\}$. We claim that the sequence $Z_1, \ldots, Z_\tau$, where $Z_t = \alpha^T(\tilde{v}_t - v_t - \mu_t)$ is a bounded martingale difference sequence. Indeed, $\mathbb{E}_r Z_0 = 0$ by the previous argument. A bound on the range of the random variables can be computed by observing

\[
|\alpha^T \tilde{v}_1| = |\alpha^T(b(b_1^\top w_t + p_t) e_t)| \leq 2b |b_1^\top w_t + p_t| \leq 2b(M + N)
\]

and

\[
|\alpha^T \mu_t| \leq 2 \|\mu_t\| \leq 2b\mu_{\text{max}} \quad |\alpha^T v_1| \leq 2bM
\]

implying

\[
|Z_t| \leq 2b(2M + N + \mu_{\text{max}}) = c.
\]

An application of Azuma-Hoeffding inequality (see e.g. Lemma A.7 in [10]) for martingale differences yields, for the fixed $\alpha$,

\[
\Pr\left(\sum_{t=1}^{\tau} Z_t > c \sqrt{2\tau \ln(2(2b)^{-1})}\right) < \delta/2^b.
\]

Having proved a statement for a fixed $\alpha$, we would like to apply the union bound to arrive at the corresponding statement for any $\alpha \in [-2,2]^b$. This is implausible as the set is uncountable. However, applying a union bound over the vertices of the hypercube $\{-2,2\}^b$ is enough. Indeed, if $|\sum_{t=1}^{\tau} Z_t| = |\alpha^T \sum_{t=1}^{\tau} (\tilde{v}_t - v_t - \mu_t)| \leq \xi$ for all vertices of $\{-2,2\}^b$, then immediately $|\sum_{t=1}^{\tau} Z_t| \leq \xi$ for any $\alpha \in [-2,2]^b$ by linearity. Thus, by union bound,

\[
\Pr\left(\bigvee_{\alpha \in [-2,2]^b} \left|\sum_{t=1}^{\tau} (\tilde{v}_t - v_t - \mu_t)\right| \leq \xi\right) \leq (2^b - 1) \delta
\]

and the statement follows by dividing by $\tau$.

With the help of Lemma 4.1, we can now analyze how the longest (or almost-longest) paths with respect to the estimated $\tilde{w}_t$’s compare to the true longest paths.

**Definition 4.1:** Define the set of $\varepsilon$-longest paths with respect to the actual delays

\[
\mathcal{S}_\varepsilon^\tau = \left\{ x \in \mathcal{P} : \frac{1}{\tau} \sum_{t=1}^{\tau} \tilde{w}_t^\top x \geq \max_{x' \in \mathcal{P}} \frac{1}{\tau} \sum_{t=1}^{\tau} \tilde{w}_t^\top x' - \varepsilon \right\}
\]

and with respect to the the estimated delays

\[
\tilde{\mathcal{S}}_\varepsilon^\tau = \left\{ x \in \mathcal{P} : \frac{1}{\tau} \sum_{t=1}^{\tau} \tilde{w}_t^\top x \geq \max_{x' \in \mathcal{P}} \frac{1}{\tau} \sum_{t=1}^{\tau} \tilde{w}_t^\top x' - \varepsilon \right\}.
\]

In particular, $\mathcal{S}_0^\tau$ is the set of longest paths.
The following Lemma makes our intuition precise: with enough trials \( t \), the set of longest paths, which we can calculate after running Algorithm 2, becomes identical to the true set of longest paths. We illustrate this point graphically in Figure 4: In a histogram of average path lengths, the set of longest paths (the right “bump”) is somewhat smeared when considering the path lengths under the estimated \( \tilde{w}_i \)'s. In other words, paths might have a slightly different average path length under the estimated and actual weights. However, we can still guarantee that this smearing becomes negligible for large enough \( t \), enabling us to locate the longest paths.

**Lemma 4.2:** For any \( \epsilon > 0 \) and for \( \xi = 2b\eta_{\text{max}} + \epsilon^{-1/2}c\sqrt{2b + 2\ln(2\delta^{-1})} \),
\[
S^\xi_t \subseteq S^{\xi+2\xi}_t \quad \text{and} \quad S^\xi_t \subseteq S^{\xi}_t
\]
with probability at least \( 1 - \delta \).

**Proof:** Let \( x \in S^\xi_t \) and \( y \in S^\xi_0 \). Suppose that we are in the \((1 - \delta)\)-probability event of Lemma 4.1. Then
\[
\frac{1}{t} \sum_{i=1}^{t} w_i^T x - \frac{1}{t} \sum_{i=1}^{t} \tilde{w}_i^T x - \xi \geq \max_{x' \in \mathcal{P}} \frac{1}{t} \sum_{i=1}^{t} \tilde{w}_i^T x' - \epsilon - \xi
\]
\[
\geq \frac{1}{t} \sum_{i=1}^{t} \tilde{w}_i^T y - \epsilon - \xi \geq \frac{1}{t} \sum_{i=1}^{t} w_i^T y - \epsilon - 2\xi
\]
\[
= \max_{x' \in \mathcal{P}} \frac{1}{t} \sum_{i=1}^{t} w_i^T x' - \epsilon - 2\xi,
\]
where the first and fourth inequalities follow by Lemma 4.1. The third inequality is by definition of maximum, and the second and fifth are by definitions of \( S^{\xi}_t \) and \( S^\xi_0 \), resp. Since the sequence of inequalities holds for all \( x \in S^\xi_t \), we conclude that \( S^\xi_t \subseteq S^{\xi+2\xi}_t \). The other direction of inclusion is proved analogously.

While the above statement is very general, we now give one interesting implication for finding a longest path under the following assumption.

**Assumption 4.1:** There exists a single path \( x^* \) that is the longest path on any round with a certain (known) margin \( \rho \):
\[
\forall x \in \mathcal{P}, \forall t, \ (x^* - x)^T w_t \geq 0
\]

Under the above margin assumption, we can, in fact, recover the longest path, as shown in the next Theorem.

**Theorem 4.2:** Suppose Assumption 4.1 holds with \( \rho > 4b\eta_{\text{max}} \). We run the Algorithm 2 for \( t = 8(\rho - 4b\eta_{\text{max}})^{-2}c^2(b + \ln(2\delta^{-1})) \) iterations.

Then with probability at least \( 1 - \delta \), Algorithm 2 outputs
\[
x^*_t := \arg \max_{x \in \mathcal{P}} \frac{1}{t} \sum_{i=1}^{t} \tilde{w}_i
\]
and \( x^*_t \) is equal to \( x^* \).

**Proof:**
Let \( x^*_t := \arg \max_{x \in \mathcal{P}, \tau} \frac{1}{\tau} \sum_{t=1}^{\tau} \tilde{w}_i \)). We claim that, with probability \( 1 - \delta \) it is equal to \( x^* \). Indeed, suppose \( x^*_t \neq x^* \). By Lemma 4.2, \( x^*_t \in S^\xi_t \subseteq S^{\xi+2\xi}_t \).

Thus,
\[
\frac{1}{\tau} \sum_{t=1}^{\tau} w_i^T x^*_t \geq \frac{1}{\tau} \sum_{t=1}^{\tau} w_i^T x^* - 2\xi
\]
leading to a contradiction whenever \( \rho \geq 2\xi = 4b\eta_{\text{max}} + \epsilon^{-1/2}c\sqrt{2b + 2\ln(2\delta^{-1})} \).

Rearranging the terms and using \( \rho - 4b\eta_{\text{max}} > 0 \), we arrive at \( \tau \geq 8(\rho - 4b\eta_{\text{max}})^{-2}c^2(b + \ln(2\delta^{-1})) \), as assumed. We conclude that with probability at least \( 1 - \delta \), \( x^*_t = x^* \) and \( \{x^*_t\} = S^\xi_t \subseteq S^{\xi+2\xi}_t \).

The following weaker assumption also has interesting implications.

**Assumption 4.2:** There exists a path \( x^* \in \mathcal{P} \) such that it is the longest path on any round
\[
\forall x \in \mathcal{P}, \forall t, \ (x^* - x)^T w_t \geq 0
\]

If, after running Algorithm 2 for enough iterations, we find all \( \epsilon \)-longest paths, Lemma 4.2 guarantees that, under Assumption 4.2, the longest path \( x^* \) is one of them with high probability. We can then use this information to test the candidate paths to find the worst-performing path over another set of iterations. We omit the details due to lack of space.

**V. Experimental Results**

**A. Implementation**

Our timing analysis tool, also called GAMETIME, operates in four stages, as described below.

1. **Extract CFG.** GAMETIME begins by extracting the control-flow graph (CFG) of the real-time task whose WCET must be estimated. This part of GAMETIME is built on top of the CIL front end for C [16]. Our CFG parameters (numbers of nodes, edges, etc.) is thus specific to the CFG representations constructed by CIL. In general, nodes correspond to the start of basic blocks of the program and edges indicate flow of control, with edges labeled by a conditional or basic block. In our experience, this phase is usually fast, taking no more than a minute for any of our benchmarks.

2. **Compute basis paths.** The next step for GAMETIME is to compute the set of basis paths and the \( B^+ \) matrix. This is done as discussed in Section IV. This phase can be somewhat time-consuming; in our experiments, the basis computation for the largest benchmark (statemate) took about 15 minutes.
3. Generate program inputs. Given the set of basis paths for the graph, GAME TIME then has to generate inputs to the program that will drive the program’s execution down that path. It does this using constraint-based test generation, by generating a constraint satisfaction problem characterizing each basis path, and then using a constraint solver based on Boolean satisfiability (SAT). This phase uses the UCLID decision procedure [17] to generate inputs for each path and creates one copy of the program for each path, with the different copies only differing in their initialization functions. For our experiments, this constraint-based test generation phase was also very quick, taking less than a minute for each benchmark. It is possible for the set of constraints for a basis path to be infeasible. In such a case, we heuristically adjust the basis to find a feasible set of paths. In all our experiments so far, the generated set of basis paths has always been feasible. Developing more systematic strategies for dealing with infeasible paths is left to future work.

4. Predict longest path. Finally, Algorithm 2 is run with the set of basis paths and their corresponding programs, along with the $B^T$ matrix. The number of iterations in the algorithm, $\tau$, depends on the mode of usage of the tool. In the experiments reported below, we used a deterministic simulator, and hence $\tau$ was set equal to $b$, since we perform one simulation per basis path. In general, $\tau$ can be pre-computed as described in Section IV or increased gradually while searching for convergence to a single longest path.

The run-time for this phase depends on the execution time of the program and the number of iterations of the loop in Algorithm 2; for our experiments, this run-time was under a minute for all benchmarks.

The estimated longest path is then executed (or simulated) several times to calculate our estimate of the WCET.

B. Benchmarks

Our benchmarks were selected from amongst those used in the WCET Challenge 2006 [18], which were drawn from the Mälardalen benchmark suite [19] and the PapaBench suite [15]. In particular, we looked for benchmarks with two features. First, rather than use artificially constructed toy benchmarks, we looked for implementations of actual real-time systems. Second, we looked for benchmarks that had several paths and were of various sizes, but which did not require automatic estimation of loop bounds. This second criterion ruled out, for example, benchmarks that compute a discrete cosine transform or did data compression, because there is usually just one path through those programs (going through several iterations of a loop), and variability in run-time usually only comes from characteristics of the data. Most benchmarks in the Mälardalen suite are of this nature.

The main characteristics of chosen benchmarks is shown in Table I. The first three benchmarks, attitude, stabilisation, and climb\_control, are tasks in the open source PapaBench software for an unmanned aerial vehicle (UAV) [15]. The last benchmark, stateate, is part of the code generated from a STATEMATE Statecharts model for an automotive window control system (single loop iteration only). Note in particular, how the number of basis paths $b$ is far less than the total number of source-sink paths in the CFG. (We are able to efficiently count the number of paths as the CFG is a DAG.) We also indicate the number of lines of code for each task; however, note that this is an imprecise metric as it includes declarations, comment lines, and blank lines – the CFG size is a more accurate representation of size.

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Size of CFG</th>
<th>Total Num. of paths</th>
<th>Num. of basis paths $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>12</td>
<td>12 $\times$ 16</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>stabilisation</td>
<td>48</td>
<td>31 $\times$ 39</td>
<td>216</td>
<td>10</td>
</tr>
<tr>
<td>climb_control</td>
<td>43</td>
<td>40 $\times$ 56</td>
<td>657</td>
<td>18</td>
</tr>
<tr>
<td>stateate</td>
<td>916</td>
<td>290 $\times$ 471</td>
<td>$7 \times 10^{16}$</td>
<td>183</td>
</tr>
</tbody>
</table>

C. Comparison using SimpleScalar Simulations

We performed experiments to compare GAME TIME against Chronos [6] as well as against testing the programs on randomly-generated inputs. WCET estimates are output in terms of the number of CPU cycles taken by the task to complete in the worst-case.

Chronos is currently the only publicly available WCET estimation tool that also participated in the WCET Challenge 2006 [18]. Chronos is built upon SimpleScalar [20], a widely-used tool for processor simulation and performance analysis. Chronos extracts a CFG from the binary of the program (compiled for MIPS using modified SimpleScalar tools), and uses a combination of dataflow analysis, integer programming, and manually constructed processor behavior models to estimate the WCET of the task.

To compare GAME TIME against Chronos, we used SimpleScalar to simulate, for each task, each of the extracted basis paths. We used the same SimpleScalar processor configuration as we did for Chronos (which is Chronos’ default configuration), specified below:

```
```

Since SimpleScalar’s execution is deterministic for a fixed processor configuration, we did not run Algorithm 2 in its entirety. Instead, we simulated each of the basis paths exactly once (factoring out the time for initialization code) and then predicted the longest path as described in Section IV. The predicted longest path was then simulated once and its execution time is reported as GAME TIME’s WCET estimate.

The random testing was done by generating initial values for each program input variable uniformly at random from its domain. For each benchmark, we generated 500 such random initializations; note that GAME TIME performs significantly fewer simulations (only as many as there are basis paths, for a maximum of 183 for the stateate benchmark).

Our results are reported in Table II. We note that the estimate of GAME TIME $T_E$ is lower than the WCET $T_C$ reported by Chronos for three out of the four benchmarks. Interestingly, $T_E > T_C$ for the stabilisation benchmark; on closer inspection, we found that this occurred mainly because the number of misses in the instruction cache was significantly underestimated by Chronos. The over-estimation by Chronos versus 19 mis-predictions on the longest path simulated by GAME TIME in SimpleScalar. In fact, the number
of branches performed in a single loop of the state code is bounded by approximately 40. (Similar overestimation by Chronos was also observed for this benchmark in the WCET Tools Challenge [18].)

We also note that GAME TIME’s estimates can be significantly higher than those generated by random testing. Moreover, GAME TIME’s predicted WCET is higher than the execution time of any of the basis paths, indicating that the basis paths taken together provide more longest path information than available from them individually.

D. Discussion

A good WCET estimation tool generates estimates that are upper bounds on the true worst-case time with low over-estimation. Evaluating the over-estimation by tools is difficult with uniform random testing, because inputs that trigger the worst-case path can easily be missed. In this context, GAME TIME offers a better alternative to bound the over-estimation by a WCET computation tool such as Chronos, as demonstrated above. We observe that GAME TIME can also be used to find timing-related bugs in real-time programs.

Importantly, GAME TIME avoids imprecisions from processor behavior analysis, sometimes generating larger times than the WCET bound generated by conservative techniques that rely on such analyses (as shown for the stabilisation benchmark). Under certain assumptions formalized in Section IV, GAME TIME is guaranteed to converge to the longest path.

A major strength of the GAME TIME approach is in its ease of applicability to a wide range of platforms. We have already applied GAME TIME on an x86-based platform (a 2.8 GHz Intel Xeon processor running Redhat Enterprise Linux) and also on the PRET processor [14]; for lack of space, we defer a discussion of these results to the full paper. Importantly, our applications of GAME TIME to these complex platforms has been done with no manual architectural modeling.

VI. CONCLUSIONS

In summary, we have presented a new, game-theoretic approach to estimating the worst-case execution (WCET) time of a software task. Our tool, GAME TIME, is measurement-based, making it easy to use on many different platforms without the need for tedious processor behavior analysis. We have presented theoretical and empirical evidence for the utility of the GAME TIME approach to timing estimation.

For future work, we note the possibility of combining GAME TIME with traditional static techniques for WCET estimation, with the goal of improving scalability and precision. We also note that our algorithm and results of Section IV are general, in that they apply to estimating longest paths in DAGs in an unpredictable environment, not just to WCET estimation for embedded software. This raises the intriguing possibility of the relevance of our algorithm to timing analysis of combinational circuits under variability. Moreover, the “longest path” need not only refer to execution time — it could also refer to other quantitative system parameters, such as power consumption. These directions appear to be worth investigating.

Acknowledgments. We are grateful to S. Arunkumar, R. Karp, E. Lee, and P. Varaiya for valuable discussions and feedback. We thank B. Seshadri, S. Juvetak, S. Jha, and the anonymous reviewers for their inputs. The first author was supported in part by NSF CAREER grant CNS-0644436, and the second author by DARPA grant FA8750-05-2-0249.

REFERENCES


TABLE II

Comparison with Chronos and random testing. Execution time estimates are in number of cycles reported by SimpleScalar. For random testing, the maximum cycle count over 500 runs is reported. The fifth column indicates the percentage over-estimation by Chronos over GAME TIME, and the last two columns indicate the maximum and minimum cycle counts for basis paths generated by GAME TIME.

<table>
<thead>
<tr>
<th>Name of Benchmark</th>
<th>Chronos WCET $T_C$</th>
<th>Random testing $T_R$</th>
<th>$T_C - T_R$</th>
<th>Basis path times $T_B$</th>
<th>$T_C - T_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude stabilisation</td>
<td>567</td>
<td>175</td>
<td>348</td>
<td>62.9</td>
<td>160</td>
</tr>
<tr>
<td>climb, control</td>
<td>1379</td>
<td>1435</td>
<td>1513</td>
<td>31.3</td>
<td>8.9</td>
</tr>
<tr>
<td>statement</td>
<td>1254</td>
<td>646</td>
<td>952</td>
<td>31.9</td>
<td>1271</td>
</tr>
<tr>
<td>8584</td>
<td>4249</td>
<td>4252</td>
<td>101.9</td>
<td>3735</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name of Benchmark</th>
<th>Chronos WCET $T_C$</th>
<th>Random testing $T_R$</th>
<th>$T_C - T_R$</th>
<th>Basis path times $T_B$</th>
<th>$T_C - T_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude stabilisation</td>
<td>567</td>
<td>175</td>
<td>348</td>
<td>62.9</td>
<td>160</td>
</tr>
<tr>
<td>climb, control</td>
<td>1379</td>
<td>1435</td>
<td>1513</td>
<td>31.3</td>
<td>8.9</td>
</tr>
<tr>
<td>statement</td>
<td>1254</td>
<td>646</td>
<td>952</td>
<td>31.9</td>
<td>1271</td>
</tr>
<tr>
<td>8584</td>
<td>4249</td>
<td>4252</td>
<td>101.9</td>
<td>3735</td>
<td></td>
</tr>
</tbody>
</table>


582