Architecture-Parametric Timing Analysis

Jan Reineke, Johannes Doerfert
Department of Computer Science
Saarland University
Saarbrücken, Germany
Email: {reineke|doerfert}@cs.uni-saarland.de

***Abstract*—Platforms are families of microarchitectures that implement the same instruction set architecture but that differ in architectural parameters, such as frequency, memory latencies, or memory sizes. The choice of these parameters influences execution time, implementation cost, and energy consumption.**

In this paper, we introduce the first general framework for *architecture-parametric timing analysis* (APTA). APTA computes an expression that bounds the worst-case execution time (WCET) of a program in terms of architectural parameters. This enables to configure a platform, at design or even at run time, in a way that is guaranteed to meet all deadlines, while minimizing implementation cost and/or energy consumption.

We demonstrate the feasibility of our approach by implementing APTA for a precision-timed (PRET) platform and by evaluating our implementation on Mälardalen benchmarks.

I. Introduction

To support a wide range of application scenarios, hardware manufacturers develop *platforms*, i.e., families of microarchitectures that implement the same instruction set architecture (ISA) and follow a common "microarchitectural template" but that differ in a number of architectural parameters. Such platforms are configurable, both statically at design time, as well as dynamically at run time. As an example, a system designer may, at design time, configure the processor frequency, the cache size, the cache associativity, the latency of main memory, the inter-connect topology, bandwidth, and arbitration policy.

At run time, for example, to trade-off performance against power consumption, techniques like dynamic resizing of buffers/caches, dynamic frequency/voltage scaling (DVFS, such as PowerNow in AMD and SpeedStep in Intel), and dynamic power management (such as ACPI), etc. have been adopted. Configuration at run time is important for building sustainable real-time embedded systems. Moreover, as processor characteristics may change over time, e.g., the maximal sustainable frequency may drop due to hardware degradation or transient faults, one should be able to reconfigure the system at run time to cope with such changes in system characteristics.

Further, there is a trend to transition from *federated architectures*, in which features are implemented on physically separated platforms, to *integrated architectures* [1], [2] that implement multiple features on a single platform. This transition has the potential to significantly reduce the amount of required resources, energy consumption, and weight, which is an important aspect in avionics. As more and more features are packed onto a single platform, integrated architectures require more powerful microarchitectures: they necessitate the use of multi- or even many-core architectures. Such architectures feature *shared resources*, such as caches, interconnect, and memory controllers. Uncontrolled sharing of these resources yields interference that can be detrimental to performance and that is extremely difficult or even impossible to predict statically. Thus, such architectures can only be used, if different tasks running simultaneously on different cores are *temporally isolated* [3], [4] from each other. This is achieved by partitioning resources in space and time. For flexibility and efficiency, the partitioning of resources is not fixed once and for all by the manufacturer, but configurable at run time. A key challenge to the efficient use of integrated architectures is thus to determine how to partition the shared resources to guarantee each application's timing constraints.

As the above observations demonstrate, today's platforms are increasingly configurable. Different configurations, i.e., different choices of parameter values, offer different tradeoffs between implementation cost, performance, and energy consumption. Thus, there is a need and an opportunity to find configurations that meet an application's performance needs, while minimizing energy consumption and implementation cost.

In the context of real-time systems, an application's performance needs are characterized by constraints on its timing behavior. Worst-case execution time (WCET) and schedulability analyses are traditionally used to verify that all timing constraints of an application are met by a given execution platform. Existing WCET and schedulability analyses, however, assume that all configuration choices of the platform have already been made, and that the configuration remains fixed at run time. Yet, to safely configure a platform at design or run time one needs to know how different configuration choices affect an application's worst-case execution time.

To this end, we introduce a framework for *architecture-parametric timing analysis (APTA)*. APTA computes an expression that bounds the WCET of a program in terms of architectural parameters. In contrast to existing WCET analyses, APTA applies to yet *unconfigured* platforms, and it reveals how an application's WCET depends on the platform's parameters. APTA can thus be a basis for an informed configuration of the platform, at design and even at run time.

To realize APTA, we follow a black-box approach, sketched in Figure 1. In this approach, we use a black-box WCET analysis to obtain bounds on the WCET of a given software binary for several specific parameter valuations. As existing WCET analyses are usually configurable w.r.t. platform parameters, the development of such a black box is not a research challenge. The bounds computed by the black box are then generalized to obtain a parametric WCET formula that applies to *all* possible parameter valuations of the platform. Two challenges arise:

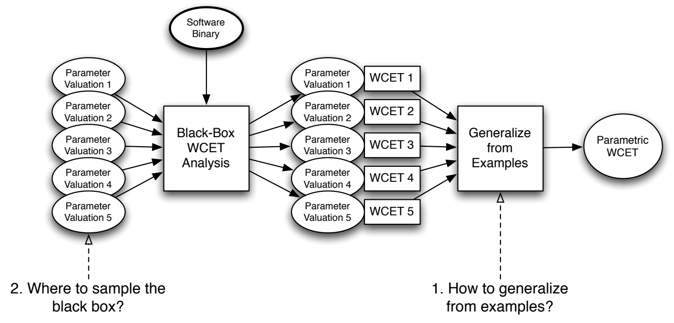

Fig. 1. Black-box approach to architecture-parametric timing analysis.

1) How to ensure soundness, i.e., how to generalize the examples provided by the black box to a correct parametric bound?
2) How to ensure precision, i.e., at which parameter valuations should the black box be sampled to obtain a parametric bound that is close to the actual WCET?

To solve the "soundness challenge", we identify sufficient conditions on the platform that enable sound generalization in Section III. Given a platform that satisfies these conditions, in Section IV, we then go on to show how to generalize from examples using *parametric linear programming*, which is introduced in Section II.

The precision of the obtained parametric bound critically depends on where the black box is sampled. We address the "precision challenge" by introducing a refinement loop in Section V that intelligently chooses parameter valuations to sample. The refinement loop can be shown to terminate *and* deliver a parametric bound that is *provably precise*. The required precision is an input to the algorithm.

As a proof of concept, we instantiate our analysis framework for a precision-timed (PRET) platform. We arrive at this platform by parameterizing the PTARM [5] microarchitecture, a predictable multi-threaded microarchitecture, which has been realized in FPGA. In particular, we make the processing frequency, the scratchpad and DRAM latencies, as well as the instruction and data scratchpad sizes parameters. We successfully apply the new analysis to a set of Mälardalen benchmarks. The proof-of-concept demonstrates that (a) the requirements for APTA can be met by actual platforms, and (b) that APTA can be effectively applied to such a platform.

To summarize, we make the following major contributions:

1) We identify sufficient conditions on the platform that enable precise and efficient parametric analysis.
2) We introduce a general framework for architecture-parametric timing analysis for any platform meeting the conditions identified in step one.
3) We instantiate the framework for a precision-timed platform and evaluate it on Mälardalen benchmarks.

Minor contributions include:

1) A black-box WCET analysis for the PTARM based on OTAWA, an open toolbox for WCET analysis.
2) Algorithms to manipulate piece-wise linear functions.

II. Background: Parametric Linear Programming

Linear programming is a mathematical method to determine the minimal (or maximal) value of a linear objective function under a set of linear constraints. *Parametric linear programming* (PLP) is an extension of linear programming in which the linear constraints defining the problem contain parameters. Parametric linear programming algorithms determine the minimal value of the linear objective function in terms of the parameters of the problem. The PLP problems that occur in this paper can be written in the following canonical form, which is a simplified form of the one considered by Feautrier [6]:

$$\begin{aligned} F(\boldsymbol{z}) = \min_{\boldsymbol{x}} \boldsymbol{c}^T \cdot \boldsymbol{x}, \\ \text{s.t. } A\boldsymbol{x} + B\boldsymbol{z} \geq \boldsymbol{d}, \\ \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{z} = (z_1, \ldots, z_n)$ is a vector of real-valued parameters, $\boldsymbol{x}$ is a vector of real-valued variables, A and B are matrices of integer-valued coefficients, $\boldsymbol{c}$, and $\boldsymbol{d}$ are vectors of integer-valued coefficients, and $\boldsymbol{0}$ is the null vector $(0, \ldots, 0)$.

For some or all parameter values the constraints may have no solutions. For convenience, we define the minimum over the empty set to be ∞. Thus, $F : Val \rightarrow (\mathbb{R} \cup \{\infty\})$ assigns a value to every valuation $\sigma \in Val = \mathbb{R}^n$ of the parameters $z_1, \ldots, z_n$.

Feautrier [6] has generalized the Simplex algorithm [7] to determine a closed-form expression ϕ with $[\![\phi]\!] = F$. In fact, the algorithm described in [6] and implemented in the library PipLib can also cope with integer variables and determine lexico-minimal solutions. Yet, the above form is sufficient for the problems considered in this paper. Closed-form solutions determined by the algorithm in [6] are called *Affine Selection Trees* (AST) and adhere to the following grammar:

$$\begin{array}{rcl} \text{AST} & ::= & \infty \mid -\infty \mid \textsc{LinearComb.} \mid \\ & & \textsc{undefined} \mid \\ & & (\textsc{LinearConst.}\ ?\ \text{AST} : \text{AST}) \\ \textsc{LinearConst.} & ::= & \textsc{LinearComb.} \geq 0 \\ \textsc{LinearComb.} & ::= & \textsc{Rational} \mid \textsc{Rational}\cdot\textsc{Variable} \mid \\ & & \textsc{LinearComb.} + \textsc{LinearComb.} \\ \textsc{Variable} & ::= & z_1 \mid z_2 \mid \ldots \end{array}$$

An example AST would be $(z_1 > 0\ ?\ 3\cdot z_1 + 5/2\cdot z_2\ :\ \infty)$. We will use ϕ and ψ as metavariables for ASTs, c as a metavariable for linear constraints, l, m as metavariables for linear combinations, and r as a metavariable for rational numbers.

An Affine Selection Tree ϕ represents a function, denoted by $[\![\phi]\!]$, with $[\![\phi]\!] : Val \rightarrow (\mathbb{R} \cup \{\infty\})$, in the expected way:

$$\begin{aligned} [\![r\cdot z_i]\!]\sigma &:= r \cdot \sigma_i \\ [\![l + m]\!]\sigma &:= [\![l]\!]\sigma + [\![m]\!]\sigma \\ [\![(l \geq 0\ ?\ \phi : \psi)]\!]\sigma &:= \begin{cases} [\![\phi]\!]\sigma & : \text{if } [\![l]\!]\sigma \geq 0 \\ [\![\psi]\!]\sigma & : \text{otherwise} \end{cases} \\ & \cdots \end{aligned}$$

where $\sigma = (\sigma_1, \ldots, \sigma_n)$.

For the example AST and the valuation $\sigma = (3, 2)$ we get $[\![(z_1 > 0\ ?\ 3 \cdot z_1 + 5/2 \cdot z_2\ :\ \infty)]\!]\sigma = [\![3 \cdot z_1 + 5/2 \cdot z_2]\!]\sigma = [\![3 \cdot z_1]\!]\sigma + [\![5/2 \cdot z_2]\!]\sigma = 9 + 5 = 14$.

III. Parameterized Timing Models

A. Formalization of Timing Models

WCET analysis requires detailed *models* of a microarchitecture's timing. A *timing model* $\mathcal{M}$ assigns an execution time $ET_{P,i} \in \mathbb{R}_{\geq 0}$ to a program P under input i[1]. In current microarchitectures, the execution time of a program may vary depending on the state of the hardware. For a timing model to be sound for a given microarchitecture, the hardware's execution times on any program and input must be bounded from above by the respective execution time assigned by the timing model. This is required for the derived WCET bounds to be sound.

Architecture-parametric timing analysis requires parameterized timing models. In a parameterized timing model, execution times additionally depend on parameters. Thus, a *parameterized timing model* $\mathcal{M}$ assigns an execution time $ET_{P,i}(\pi_1, \ldots, \pi_m) \in \mathbb{R}_{\geq 0}$ to a program P under input i and parameter valuation $(\pi_1, \ldots, \pi_m) \in \mathbb{R}^m_{\geq 0}$. Parameters may determine the latencies of classes of instructions, the sizes and access latencies of different memories, such as scratchpads or caches, the latencies of input/output operations, etc.

Traditional WCET analysis is based on unparameterized timing models of particular hardware implementations of ISAs. It determines bounds on the execution time of a program under all legal inputs I, i.e., bounds on its worst-case execution time:

$$WCET_P := \max_{i \in I} ET_{P,i}.$$

Parameterized timing models induce a parametric WCET:

$$WCET_P(\pi_1, \ldots, \pi_m) := \max_{i \in I} ET_{P,i}(\pi_1, \ldots, \pi_m)$$

The goal of parametric timing analysis is to compute an expression ϕ that bounds the WCET of a program in terms of the parameters of a timing model:

$$\forall (\pi_1, \ldots, \pi_m) \in \mathbb{R}^m_{\geq 0} : \quad WCET_P(\pi_1, \ldots, \pi_m) \leq [\![\phi]\!](\pi_1, \ldots, \pi_m).$$

Such a formula should both be precise, i.e., it should be close to the actual WCET, and it should be efficiently evaluable, i.e., a large SMT formula that still needs to be solved would *not* be an adequate solution.

B. Sufficient Conditions for Precise and Efficient Analysis

Precise and efficient parametric timing analysis is not possible for arbitrary parameterized timing models. It becomes feasible only if we restrict in some way how execution times may depend on parameters. We have identified two kinds of parameters that admit parametric analysis:

1) *Linear parameters*, where part of the execution time depends linearly on each parameter. We denote linear parameters by $\lambda_1, \lambda_2, \ldots \in \mathbb{R}_{\geq 0}$.
2) *Monotone parameters*, where part of the execution time is monotone in the value of the parameter. We denote monotone parameters by $\mu_1, \mu_2, \ldots \in \mathbb{R}_{\geq 0}$.

We define parameterized timing models with linear parameters $\lambda_1, \ldots, \lambda_m$ to be timing models that can be written in the following way as a linear combination of functions $f_j(P, i) \in \mathbb{R}_{\geq 0}$ that capture how often different events occur during execution:

$$ET_{P,i}(\lambda_1, \ldots, \lambda_m) = \sum_{j=1}^{m} \lambda_j \cdot f_j(P, i).$$

Linear parameters can model properties such as the latencies of the different levels of a memory hierarchy, bus transmission delays, and the processor's cycle time, i.e., the inverse of its frequency. For these examples, the frequency functions $f_j(P, i)$ would determine the number of accesses to the different levels of the memory hierarchy, the number of bus transmissions, and the number of instructions being executed, respectively.

The frequencies, e.g., the number of accesses to the scratchpad memory or to main memory, may in turn depend on the scratchpad's size, which can be modeled by a *monotone* parameter. Thus, we allow each of the frequency functions f_j to depend on one or more monotone parameters. This leads to parameterized timing models that can be decomposed in the following way, where all f_j are monotone or antimonotone in $\mu_1, \ldots, \mu_n$:

$$ET_{P,i}(\lambda_1, \ldots, \lambda_m, \mu_1, \ldots, \mu_n) = \sum_{j=1}^{m} \lambda_j \cdot f_j(P, i, \mu_1, \ldots, \mu_n) \quad (1)$$

For example, in a memory hierarchy, μ_l may determine the size of the l^{th} memory, $f_l(P, i, \ldots, \mu_l, \ldots)$ the number of accesses to the l^{th} memory, and λ_l the memory's latency.

In the following, we develop a parametric timing analysis method for any timing model that can be decomposed into monotone functions f_j as in Equation (1). For simplicity but w.l.o.g., we assume the frequency functions to be antimonotone in their parameters, i.e., the frequencies decrease with increasing parameter values, which is natural if parameters encode sizes of memories such as caches or scratchpads.

C. Properties of Parameterized Timing Models

Our analysis approach makes use of a number of properties of such timing models that we discuss in the following. For a timing model $\mathcal{M}$ with linear parameters only, $ET_{P,i} : \mathbb{R}^m_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a linear map. Therefore it can be represented by the images of any set of parameter vectors that forms a spanning set of the parameter vector space. For example:

$$\begin{aligned} ET_{P,i}(\lambda_1, \lambda_2) &= \lambda_1 ET_{P,i}(1, 0) + \lambda_2 ET_{P,i}(0, 1) && (2) \\ &= \lambda_1 ET_{P,i}(1, 1) + (\lambda_2 - \lambda_1) ET_{P,i}(0, 1) && (3) \end{aligned}$$

Both $\{(1, 0), (0, 1)\}$ and $\{(1, 1), (0, 1)\}$ are spanning sets of the vector space $\mathbb{R}^2_{\geq 0}$.

For $1 \leq j \leq k$, let $(\lambda^j_1, \ldots, \lambda^j_m) \in \mathbb{R}^m$ be parameter vectors, and $w_j \in \mathbb{R}$ be weights. Then, $ET_{P,i}(\lambda_1, \ldots, \lambda_m)$ can be approximated from above as follows:

$$ET_{P,i}(\lambda_1, \ldots, \lambda_m) \leq \sum_{j=1}^{k} w_j ET_{P,i}(\lambda^j_1, \ldots, \lambda^j_m), \quad (4)$$

if $\lambda_i \leq \sum_{j=1}^{k} w_j \lambda^j_i$ for all $i, 1 \leq i \leq m$.

[1] We assume that programs of interest terminate under all considered inputs.

As an example, consider a timing model with two linear parameters that encode the cache latency and the main memory latency, respectively. Then, $ET_{P,i}(1,100) = 1 \cdot ET_{P,i}(1,0) + 100 \cdot ET_{P,i}(0,1)$ and $ET_{P,i}(1,100) \leq 10 \cdot ET_{P,i}(5,10)$, as 1 and 100 are less than or equal to $10 \cdot 5$ and $10 \cdot 10$, respectively. In other words, if the execution time assuming a cache latency of 5 and a main memory latency of 10 is $ET_{P,i}(5,10)$, then we can approximate the execution time $ET_{P,i}(1,100)$ for a cache latency of 1 and a main memory latency of 100 by $10 \cdot ET_{P,i}(5,10)$.

While $WCET_P$ itself is not necessarily a linear map, it can still be approximated in a similar way:

$$\begin{aligned}
WCET_P(\lambda_1,\dots,\lambda_m) &\overset{\substack{\text{Def.}\\ WCET_P}}{=} \max_{i\in I} ET_{P,i}(\lambda_1,\dots,\lambda_m) && (5)\\
&\overset{\substack{\text{Inequation}\\ (4)}}{\leq} \max_{i\in I} \sum_{j=1}^{k} w_j ET_{P,i}(\lambda_1^j,\dots,\lambda_m^j) && (6)\\
&\overset{\forall j.w_j\geq 0}{\leq} \sum_{j=1}^{k} w_j \max_{i\in I} ET_{P,i}(\lambda_1^j,\dots,\lambda_m^j) && (7)\\
&\overset{\substack{\text{Def.}\\ WCET_P}}{=} \sum_{j=1}^{k} w_j WCET_P(\lambda_1^j,\dots,\lambda_m^j) && (8)
\end{aligned}$$

if $\lambda_i \leq \sum_{j=1}^{k} w_j \lambda_i^j$ for all $i, 1 \leq i \leq m$ and $w_j \geq 0$ for all $j, 1 \leq j \leq k$.

The main difference here is that all weights w_j need to be non-negative (Inequation 7). Intuitively, as WCETs are upper bounds on execution times, we cannot subtract them from each other, and still be sure to obtain upper bounds on the WCET for other parameter values.

For timing models with antimonotone parameters we have:

$$\begin{aligned}
ET_{P,i}(\mu_1,\dots,\mu_n) &\leq ET_{P,i}(\mu_1',\dots,\mu_n'), && (9)\\
WCET_P(\mu_1,\dots,\mu_n) &\leq WCET_P(\mu_1',\dots,\mu_n'), && (10)
\end{aligned}$$

if $\mu_i \geq \mu_i'$ for all $i, 1 \leq i \leq n$.

Finally, timing models that have both linear and monotone parameters have the following property which follows from Equations (5)-(8) and Inequation (10):

Theorem 1 (WCET Bounds based on WCET Samples):

$$\begin{aligned}
&WCET_P(\lambda_1,\dots,\lambda_m,\mu_1,\dots,\mu_n)\\
&\leq \sum_{j=1}^{k} w_j WCET_P(\lambda_1^j,\dots,\lambda_m^j,\mu_1^j,\dots,\mu_n^j), \quad (11)
\end{aligned}$$

if the weights w_j are non-negative, $\lambda_i \leq \sum_{j=1}^{k} w_j \lambda_i^j$, and $\mu_i \geq \mu_i^j$ for all i and j.

This theorem shows how the WCET for parameter valuation $\lambda_1,\dots,\lambda_m,\mu_1,\dots,\mu_n$ can be bounded using the WCETs for other parameter valuations.

IV. From Samples to Parametric WCET Bounds: How to Generalize from Examples

Theorem 1 shows how to bound the WCET for *any* parameter valuation using WCETs for a set of parameter valuations. While the theorem is in terms of the actual WCETs, the black-box WCET analysis, modeled by the function BB_P, will in general only compute upper approximations of these. In the following, let $\boldsymbol{\lambda}$ denote $\lambda_1,\dots,\lambda_m$ and let $\boldsymbol{\mu}$ denote $\mu_1,\dots,\mu_n$. A black-box WCET analysis BB_P is *sound*, if

$$\forall \boldsymbol{\lambda},\boldsymbol{\mu} : WCET_P(\boldsymbol{\lambda},\boldsymbol{\mu}) \leq BB_P(\boldsymbol{\lambda},\boldsymbol{\mu}).$$

Inequation (11) also holds for such approximations, and we get:

$$WCET_P(\boldsymbol{\lambda},\boldsymbol{\mu}) \leq \sum_{j=1}^{k} w_j BB_P(\boldsymbol{\lambda}^j,\boldsymbol{\mu}^j), \quad (12)$$

if the weights w_j are non-negative, $\lambda_i \leq \sum_{j=1}^{k} w_j \lambda_i^j$, and $\mu_i \geq \mu_i^j$ for all i and j.

Given a set of black-box samples $S = \{(e_j,(\boldsymbol{\lambda}^j,\boldsymbol{\mu}^j)) \mid \forall j : 1 \leq j \leq |S|, e_j = BB_P(\boldsymbol{\lambda}^j,\boldsymbol{\mu}^j)\}$ there may be several ways of bounding $WCET_P(\boldsymbol{\lambda},\boldsymbol{\mu})$ using Inequation (12). Consider for example the set of samples $S = \{(5,(1,0)),(3,(0,1)),(7,(1,1))\}$, where both parameters are linear. Again, the two parameters in this example might model cache and main memory latencies. Then, the sample $(5,(1,0))$ indicates that at most 5 cache hits may occur during any program execution. Similarly, $(3,(0,1))$ indicates that at most 3 accesses to main memory may occur. However, the maxima for the two parameters might not occur on the same path through the program, and so it is possible that any program path contains at most 7 memory accesses (cache plus main memory), as indicated by the third sample $(7,(1,1))$. Then, $WCET_P(1,10)$ can be bounded by $1 \cdot 5 + 10 \cdot 3 = 35$ because $1 \cdot (1,0) + 10 \cdot (0,1) \geq (1,10)$, but it can also be bounded by $1 \cdot 7 + 9 \cdot 3 = 34$ because $1 \cdot (1,1) + 9 \cdot (0,1) \geq (1,10)$.

Naturally, the goal is to compute the *best* bound possible given the set of samples. The best bound $UB_S(\boldsymbol{\lambda},\boldsymbol{\mu})$ of the WCET in terms of the parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ for the set of samples S is determined by the following parametric program:

$$\begin{aligned}
UB_S(\boldsymbol{\lambda},\boldsymbol{\mu}) := \min_{\boldsymbol{w}} \quad & \sum_{j=1}^{|S|} w_j e_j,\\
\text{s.t.} \quad & \forall i : \lambda_i \leq \sum_{j=1}^{|S|} w_j \lambda_i^j, \qquad (13)\\
& \forall j : w_j \geq 0,\\
& \forall j : (\exists i : \mu_i < \mu_i^j) \rightarrow w_j = 0,
\end{aligned}$$

where $\boldsymbol{\lambda} = \lambda_1,\dots,\lambda_m$ and $\boldsymbol{\mu} = \mu_1,\dots,\mu_n$ are parameters for all i, and w_j are variables for $1 \leq j \leq |S|$.

The constraints of the parametric program correspond to the conditions of Inequation (12), and the objective function corresponds to the bound. So the program determines weights that meets the constraints while minimizing the bound.

Without the final constraint on the monotone parameters this is a parametric linear program that can be solved using PipLib [6]. A naive approach to resolve this obstacle is to construct a parametric linear program for every subset S' of the set of WCET samples S, and to symbolically compute the minimum of the analysis results of all programs. These subsets correspond to all possible choices of permitting positive weights w_j. For each such subset S', one can fix $w_j = 0$ if $s_i \notin S'$. Then, the final constraint on the monotone parameters, either becomes trivially true and can be eliminated, or can be simplified to $\forall j, i : \mu_i \geq \mu_i^j$, and we arrive at a parametric linear program (PLP).

We can do better, as some subsets are dominated by others. Consider two subsets S' and S'' of S. If $\max_{s_k \in S'} \mu_j^k = \max_{s_k \in S''} \mu_j^k$ for all j, then the PLP for $S' \cup S''$ will impose the same constraints on the monotone parameters as the PLPs for S' and S'', but it will be less constrained w.r.t. the weights w_j. Thus, the solution, $\phi_{S' \cup S''}$ for $S' \cup S''$ will be less than or equal to the solutions for both S' and S'', i.e., $[\![\phi_{S' \cup S''}]\!](\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \min\{[\![\phi_{S'}]\!](\boldsymbol{\lambda}, \boldsymbol{\mu}), [\![\phi_{S''}]\!](\boldsymbol{\lambda}, \boldsymbol{\mu})\}$ for all $\boldsymbol{\lambda}, \boldsymbol{\mu}$. Thus, we only consider the following set $\mathcal{S}$ of subsets of S:

$$\mathcal{S} = \{S' \subseteq S \mid \forall s_i \in S \setminus S' : \exists j \text{ s.t. } \mu_j^i > \max_{s_k \in S'} \mu_j^k\}.$$

There can be at most $|S|^m$ such subsets, where m is the number of monotone parameters, which may be considerably fewer than the $2^{|S|}$ subsets of S.

Algorithm 1 shows the resulting parametric timing analysis algorithm. The algorithm successively constructs and solves PLPs for all elements of $\mathcal{S}$ following Equation 13. It incrementally computes an affine selection tree (AST) representing the pointwise minimum of the functions represented by the ASTs for each element of $\mathcal{S}$. As the symbolic minimum algorithm *min*, discussed below, may incur a quadratic blow-up in the size of the AST, a reduction operation *reduce*, also discussed below, is required to keep the AST compact.

a) Symbolic Computation of Pointwise Minima: Given two ASTs ϕ and ψ, the following recursive procedure computes an AST $\mathit{min}(\phi, \psi)$, such that $[\![\mathit{min}(\phi, \psi)]\!]\sigma = \min\{[\![\phi]\!]\sigma, [\![\psi]\!]\sigma\}$ for all valuations σ:

$$\begin{aligned}
\mathit{min}((c \ ? \ \phi_1 : \phi_2), \psi) &= (c \ ? \ \mathit{min}(\phi_1, \psi) : \mathit{min}(\phi_2, \psi)) \\
\mathit{min}(\phi, (c \ ? \ \psi_1 : \psi_2)) &= (c \ ? \ \mathit{min}(\phi, \psi_1) : \mathit{min}(\phi, \psi_2)) \\
\mathit{min}(\phi, \infty) &= \phi \\
\mathit{min}(\infty, \psi) &= \psi \\
\mathit{min}(l, m) &= (l - m \geq 0 \ ? \ m : l)
\end{aligned}$$

b) Reduction of Affine Selection Trees: The algorithm described above may incur a quadratic blow-up in the size of the AST. To counter this blowup, we have devised the following algorithm, which reduces the size of an AST without changing the function it represents. It does so by identifying paths through the AST that correspond to unsatisfiable sets of constraints, and by identifying subtrees that are syntactically equal. The second parameter of *reduce* is used to collect all the constraints on the path from the root.

$$\begin{aligned}
\mathit{reduce}(\phi) &= \mathit{reduce}(\phi, \bigwedge_{x \in \{\boldsymbol{\lambda}, \boldsymbol{\mu}\}} x \geq 0) \\
\mathit{reduce}(\infty, p) &= \infty \\
\mathit{reduce}(l, p) &= l
\end{aligned}$$

$$\mathit{reduce}((c \ ? \ \phi : \psi), p) = \begin{cases} \phi' & : \text{if } p \wedge \neg c \text{ is unsatisfiable} \vee \phi' = \psi' \\ \psi' & : \text{if } p \wedge c \text{ is unsatisfiable} \\ (c \ ? \ \phi' : \psi') & : \text{otherwise} \end{cases}$$

where $\phi' = \mathit{reduce}(\phi, p \wedge c)$ and $\psi' = \mathit{reduce}(\psi, p \wedge \neg c)$. All parameters are assumed to be greater or equal to zero, which explains the first line. We check the satisfiability of the linear-arithmetic constraints using YICES [8]. For efficiency our implementation actually intertwines *min* and *reduce*.

Algorithm 1: Parametric WCET Bounds

```
Input: Set of WCET Samples $S$.
Output: Parametric upper bound on the WCET as an
        Affine Selection Tree $\phi$ with $[\![\phi]\!] = \mathit{UB}_S$.
1 begin
2     $\phi \leftarrow \infty$
3     $\mathcal{S} \leftarrow \{S' \subseteq S \mid \forall s_i \in S \setminus S' :$
4             $\exists j \text{ s.t. } \mu_j^i > \max_{s_k \in S'} \mu_j^k\}$
5     foreach $T \in \mathcal{S}$ do
6         $P_T \leftarrow$ construct-parametric-linear-program($T$)
7         $\phi_T \leftarrow \mathit{solve}(P_T)$ //call of PIPLIB library
8         $\phi \leftarrow \mathit{reduce}(\mathit{min}(\phi, \phi_T))$
9     return $\phi$
```

V. Parametric WCET Bounds with Precision Guarantees

A. Problem Formulation

In the previous section we have seen how to soundly generalize a set of samples S to a parametric WCET bound UB_S. However, the precision of the obtained parametric bound depends strongly on the given set of samples. In this section, we tackle the problem of obtaining a set of samples S, such that the resulting parametric WCET bound UB_S is *provably precise*. Ideally, we would like to determine a small set S such that

$$\forall \boldsymbol{\lambda}, \boldsymbol{\mu} : \mathit{WCET}_P(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathit{UB}_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathit{BB}_P(\boldsymbol{\lambda}, \boldsymbol{\mu}).$$

Such a set would yield a perfectly precise parametric bound with no overestimation relative to the black box. However, for an arbitrary black box this may require an infinite set of samples S. Therefore, we introduce a parameter $\epsilon \in \mathbb{R}_{>0}$ and ask for a set of samples S, such that

$$\forall \boldsymbol{\lambda}, \boldsymbol{\mu} : \mathit{WCET}_P(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathit{UB}_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) < \mathit{BB}_P(\boldsymbol{\lambda}, \boldsymbol{\mu}) + \epsilon. \quad (14)$$

In other words, we are asking for a set of samples, such that the induced parametric bound never overestimates the black box ϵ or more.

While a finite such set exists if BB_P has a finite number of discontinuities, we have no way of locating these discontinuities exactly by sampling BB_P if the monotone parameters are from a continuous domain. By definition BB_P is continuous in the linear parameters $\boldsymbol{\lambda}$, but it may be arbitrarily discontinuous in $\boldsymbol{\mu}$. Therefore, to make the problem solvable, we introduce a second parameter $\boldsymbol{\tau} \in \mathbb{R}^n_{>0}$. The goal is then to find a set of samples S, such that

$$\begin{aligned}
&\forall \boldsymbol{\lambda} \in \mathbb{R}^m_{\geq 0}, \boldsymbol{\mu} \in \mathbb{R}^n_{\geq 0}, \boldsymbol{\lambda} \leq \boldsymbol{\lambda}^{\max}, \boldsymbol{\mu} \leq \boldsymbol{\mu}^{\max} : \\
&\quad \mathit{WCET}_P(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathit{UB}_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) < \mathit{BB}_P(\boldsymbol{\lambda}, \boldsymbol{\mu} - \boldsymbol{\tau}) + \epsilon, \quad (15)
\end{aligned}$$

where $\boldsymbol{\lambda}^{\max}$ and $\boldsymbol{\mu}^{\max}$ are platform-defined maximal values for its parameters. Note that the parameter $\boldsymbol{\tau}$ must be strictly positive. This way it is sufficient to find parameter valuations to sample that are within a distance of $\boldsymbol{\tau}$ of discontinuities.

B. Refinement Approach

We follow a refinement approach, in which the sought-after set of samples S is constructed incrementally.

Given an assumption about the black box that we will detail in the following section, in addition to upper bounds, we can

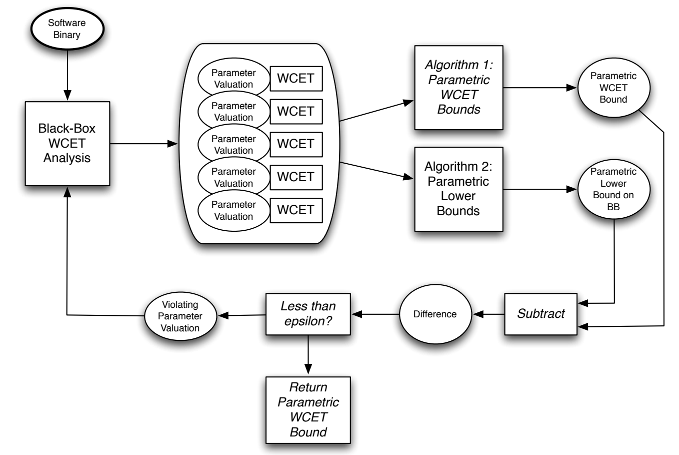

Fig. 2. Refinement flow for architecture-parametric timing analysis.

also determine *lower bounds* LB_S on the black box based on a set of samples S, such that

$$\forall \boldsymbol{\lambda}, \boldsymbol{\mu} : LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu}). \tag{16}$$

Thus, if $UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) < LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu} - \boldsymbol{\tau}) + \epsilon$ then $UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) < BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu} - \boldsymbol{\tau}) + \epsilon$ follows. So, the goal is to construct a set of samples S, such that

$$\forall \boldsymbol{\lambda} \in \mathbb{R}^m_{\geq 0}, \boldsymbol{\mu} \in \mathbb{R}^n_{\geq 0}, \boldsymbol{\lambda} \leq \boldsymbol{\lambda}^{\max}, \boldsymbol{\mu} \leq \boldsymbol{\mu}^{\max} : \\ UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) < LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu} - \boldsymbol{\tau}) + \epsilon. \tag{17}$$

Such a set S also satisfies Equation (15).

Starting with an arbitrary set of samples S, we incrementally add samples in the following way, illustrated in Figure 2:

1) We determine a parameter valuation $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ that maximizes $UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu} + \frac{\boldsymbol{\tau}}{2}) - LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu} - \frac{\boldsymbol{\tau}}{2})$. Note, that the maximal difference is equal to the maximal difference between $UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu})$ and $LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu} - \boldsymbol{\tau})$.
2) If the difference between upper and lower bound is less than ϵ, we terminate, and S satisfies Equation (15).
3) Otherwise, we add $(BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu}), (\boldsymbol{\lambda}, \boldsymbol{\mu}))$ to S and continue with step 1.

After adding $(BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu}), (\boldsymbol{\lambda}, \boldsymbol{\mu}))$ to S, the lower and upper bounds coincide with the black box at $(\boldsymbol{\lambda}, \boldsymbol{\mu})$: $LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) = UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) = BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu})$. Also, it can be shown that in a neighborhood around $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, the lower and upper bounds differ by less than ϵ. Thus, after adding $(BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu}), (\boldsymbol{\lambda}, \boldsymbol{\mu}))$ to S, no valuation in its neighborhood will have to be sampled anymore. This guarantees termination.

Before describing the refinement algorithm in more detail, let us explain how to obtain lower bounds on the black box.

C. Computing Lower Bounds on the Black Box

To compute lower bounds on the black box, we make the assumption that the black box is *reasonable*. In order to formally define what it means for a black box to be reasonable, first consider the following definition of redundancy of samples:

Definition 1 (Strict Redundancy of Samples): A sample $(e, (\boldsymbol{\lambda}, \boldsymbol{\mu}))$ is *strictly redundant* w.r.t. to a set of samples S if $UB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) < e$. We call a set of samples S *strictly redundant* if there is a sample $s \in S$ that is strictly redundant w.r.t. S.

Algorithm 2: Parametric Lower Bounds on Black Box

```
  Input: Set of WCET Samples $S$.
  Output: Parametric lower bound on black box as an
          Affine Selection Tree $\phi$ with $[\![\phi]\!] = LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu})$.
1 begin
2     $\phi \leftarrow 0$
3     foreach $s' \in S$ do
4         //following Equation (18):
5         $P_{S,s'} \leftarrow$ construct-param.-linear-program$(S, s')$
6         $\phi_{S,s'} \leftarrow solve(P_{S,s'})$ //call of PIPLIB library
7         $\phi_{S,s'} \leftarrow \phi_{S,s'}[\text{UNDEFINED} \mapsto -\infty]$
8         $\phi \leftarrow reduce(max(\phi, \phi_{S,s'}))$
9     return $\phi$
```

In other words, a better estimate of $WCET(\boldsymbol{\lambda}, \boldsymbol{\mu})$ than e can be derived based on the samples in S.

As an example, consider again the scenario of two linear parameters, modeling the latencies of the cache and the main memory, respectively. The set of samples $S = \{(5, (1, 0)), (3, (0, 1)), (9, (1, 1))\}$ is strictly redundant, as the third sample $(9, (1, 1))$ is strictly redundant w.r.t. to first two samples: the upper bound $UB_S(\lambda_1, \lambda_2)$ is $5 \cdot \lambda_1 + 3 \cdot \lambda_2$, based only on the first two samples. $UB_S(1, 1) = 8$ yields a better bound on $WCET(1, 1)$ than the third sample.

We consider a black box to be reasonable, if it *never* produces strictly redundant sets of samples:

Definition 2 (Reasonable Black Box): A black box BB is *reasonable* if there is no program P and set of parameter valuations V such that $S = \{(BB_P(\boldsymbol{\lambda}, \boldsymbol{\mu}), (\boldsymbol{\lambda}, \boldsymbol{\mu})) \mid (\boldsymbol{\lambda}, \boldsymbol{\mu}) \in V\}$ is strictly redundant.

Reasonable black boxes allow us to derive lower bounds from the samples that we have already made: for each parameter valuation we can determine the smallest possible value that would *not* make *any* existing sample redundant. The following formula determines the lowest value e for a given parameter valuation that does not make the sample $(e', (\boldsymbol{\lambda}', \boldsymbol{\mu}')) \in S$ redundant w.r.t. $S \cup \{(e, (\boldsymbol{\lambda}, \boldsymbol{\mu}))\}$:

$$LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}(\boldsymbol{\lambda}, \boldsymbol{\mu}) := \min_e \quad e, \\ \text{s.t.} \quad e' \leq UB_{S \cup \{(e,(\boldsymbol{\lambda},\boldsymbol{\mu}))\}}(\boldsymbol{\lambda}', \boldsymbol{\mu}').$$

Each sample $(e', (\boldsymbol{\lambda}', \boldsymbol{\mu}')) \in S$ thus gives rise to a lower bound on the black box. These lower bounds can be combined to arrive at $LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu})$, the smallest value that does not make *any* sample in S redundant:

$$LB_S(\boldsymbol{\lambda}, \boldsymbol{\mu}) := \max_{(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}')) \in S} LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}(\boldsymbol{\lambda}, \boldsymbol{\mu}).$$

Given an AST $\phi_{S,s}$ for each $s \in S$ that represents $LB_{S,s}$, i.e., $[\![\phi_{S,s}]\!] = LB_{S,s}$, we can compute an AST ϕ that represents LB_S by symbolically computing the pointwise maximum of all $\phi_{S,s}$. This can be done analogously to the computation of pointwise minima discussed in Section IV.

How do we compute the AST $\phi_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}$? First, note that if $\boldsymbol{\mu}' \not\geq \boldsymbol{\mu}$, then $LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}(\boldsymbol{\lambda}, \boldsymbol{\mu}) = -\infty$, because then $UB_{S \cup \{(e,(\boldsymbol{\lambda},\boldsymbol{\mu}))\}}(\boldsymbol{\lambda}', \boldsymbol{\mu}')$ is independent of e.

Now, consider the case that $\boldsymbol{\mu}' \geq \boldsymbol{\mu}$. Observe that $UB_{S\cup\{(e,(\boldsymbol{\lambda},\boldsymbol{\mu}))\}}$ is monotone and continuous in e. Thus:

$$\begin{aligned} LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}(\boldsymbol{\lambda},\boldsymbol{\mu}) :=& \min_e \quad e, \\ & \text{s.t.} \quad e' \leq UB_{S\cup\{(e,(\boldsymbol{\lambda},\boldsymbol{\mu}))\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}') \\ =& \max_e \quad e, \\ & \text{s.t.} \quad UB_{S\cup\{(e,(\boldsymbol{\lambda},\boldsymbol{\mu}))\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}') \leq e' \end{aligned}$$

By plugging Program 13 for UB_S into the formula for $LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}$ and soundly eliminating the inner minimum, we arrive at the following parametric program:

$$\begin{aligned} LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}(\boldsymbol{\lambda},\boldsymbol{\mu}) :=& \max_{e,w,\boldsymbol{w}} e, \\ \text{s.t.} \quad & w{\cdot}e + \sum_{j=1}^{|S|} w_j{\cdot}e_j \leq e', \\ & \forall i : \lambda_i' \leq w{\cdot}\lambda_i + \sum_{j=1}^{|S|} w_j{\cdot}\lambda_i^j, \\ & w \geq 0, \forall j : w_j \geq 0, \\ & \forall j : (\exists i : \mu_i' < \mu_i^j) \rightarrow w_j = 0, \\ & (\exists i : \mu_i' < \mu_i) \rightarrow w = 0. \end{aligned} \quad (18)$$

The parameters of this program are $\boldsymbol{\lambda} = \lambda_1,\ldots,\lambda_m$ and $\boldsymbol{\mu} = \mu_1,\ldots,\mu_n$, while e, w, and $\boldsymbol{w} = w_1,\ldots,w_{|S|}$ are variables. Finally, for all i and j, λ_i^j, μ_i^j and λ_i', μ_i' are constants defined by the set of samples S and $(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))$.

Depending on the values of the constants μ_i' and μ_i^j, the second to last constraint $\forall j : (\exists i : \mu_i' < \mu_i^j) \rightarrow w_j = 0$ can either be eliminated or simplified to $w_j = 0$. Due to our assumption that $\boldsymbol{\mu}' \geq \boldsymbol{\mu}$, the last constraint can be sharpened to $\boldsymbol{\mu}' \geq \boldsymbol{\mu} \equiv \forall i : \mu_i' \geq \mu_i$.

After these simplifications, all constraints are linear and the program can be solved using PIPLIB. Algorithm 2 shows the procedure to compute parametric lower bounds on reasonable black boxes. The AST computed in line 6 is UNDEFINED for $\boldsymbol{\mu}' \not\geq \boldsymbol{\mu}$ due to the final constraint discussed in the previous paragraph. In these cases, $LB_{S,(e',(\boldsymbol{\lambda}',\boldsymbol{\mu}'))}(\boldsymbol{\lambda},\boldsymbol{\mu}) = -\infty$. Thus, in line 7 undefined portions of $\phi_{S,s'}$ are modified to $-\infty$. The algorithm to compute pointwise minima of ASTs described in Section IV can easily be adapted to compute pointwise maxima, used in line 8 of the algorithm.

We have shown how to compute lower bounds on the black box provided the black box is reasonable. This may appear to be a restriction, however, it really is not, for two reasons: 1. We believe that most black boxes are naturally reasonable. This is the case for the WCET analysis of the parameterized PTARM, which we discuss in Section VI-C. 2. Non-reasonable black boxes can be wrapped in a "reasonabilizer" that returns the minimum of the non-reasonable black box $BB(\boldsymbol{\lambda},\boldsymbol{\mu})$ and $UB_S(\boldsymbol{\lambda},\boldsymbol{\mu})$, where S is the set of parameter valuations that have been sampled before.

D. Putting It All Together

Algorithm 3 combines the algorithms to compute upper and lower bounds to successively generate a set of samples S and an AST $\overline{\phi}$ with $[\![\overline{\phi}]\!] = UB_S$ that satisfy Equation (15).

Algorithm 3: Parametric Timing Analysis with Precision Guarantees

```
Input: Reasonable Black Box WCET Analysis $BB$.
Bounds on parameter values $\boldsymbol{\lambda}^{\max} \in \mathbb{R}^m_{\geq 0}, \boldsymbol{\mu}^{\max} \in \mathbb{R}^n_{\geq 0}$.
Precision requirements $\boldsymbol{\tau} \in \mathbb{R}^n_{>0}$, $\epsilon \in \mathbb{R}_{>0}$.
Output: Parametric WCET bound as an AST $\overline{\phi}$ and set
        of samples $S$, such that $[\![\overline{\phi}]\!] = UB_S$, and
   $\forall \boldsymbol{\lambda} \in \mathbb{R}^m_{\geq 0}, \boldsymbol{\mu} \in \mathbb{R}^n_{\geq 0}, \boldsymbol{\lambda} \leq \boldsymbol{\lambda}^{\max}, \boldsymbol{\mu} \leq \boldsymbol{\mu}^{\max}$ :
   $WCET_P(\boldsymbol{\lambda},\boldsymbol{\mu}) \leq UB_S(\boldsymbol{\lambda},\boldsymbol{\mu}) < BB_P(\boldsymbol{\lambda},\boldsymbol{\mu}-\boldsymbol{\tau}) + \epsilon$.

1  begin
2    $V \leftarrow \{(\mathbf{1},\mathbf{0}),(\mathbf{1},\boldsymbol{\mu}^{\max})$
3        $\mid \mathbf{1} = \overbrace{(1,\ldots,1)}^{m}, \mathbf{0} = \overbrace{(0,\ldots,0)}^{n}\}$
4    $S \leftarrow \{(BB(\sigma),\sigma) \mid \sigma \in V\}$
5    $\boldsymbol{\tau}' \leftarrow \boldsymbol{\tau}_{\text{loose}}$
6    $\epsilon' \leftarrow \epsilon_{\text{loose}}$
7    while $\epsilon' \geq \epsilon$ do
8      while $\boldsymbol{\tau}' \geq \boldsymbol{\tau}$ do
9        for $i \leftarrow 1$; $i \leq n$; $i \leftarrow i+1$ do
10         while true do
11           $\overline{\phi} \leftarrow$ compute-upper-bound($S$)
12           $\underline{\phi} \leftarrow$ compute-lower-bound($S$)
13           $\phi_{\text{diff}} \leftarrow sub(trans(\overline{\phi}, \frac{\boldsymbol{\tau}'}{2}), trans(\underline{\phi}, -\frac{\boldsymbol{\tau}'}{2}))$
14           $(\delta,(\boldsymbol{\lambda}^{\delta},\boldsymbol{\mu}^{\delta})) \leftarrow max(\phi_{\text{diff}}, \boldsymbol{\lambda}^{\max}, \boldsymbol{\mu}^{\max})$
15           if $\delta > \epsilon'$ then
16             $S \leftarrow S \cup \{(BB(\boldsymbol{\lambda}^{\delta},\boldsymbol{\mu}^{\delta}),(\boldsymbol{\lambda}^{\delta},\boldsymbol{\mu}^{\delta}))\}$
17           else
18             break
19         $\tau_i' \leftarrow \frac{1}{2}{\cdot}\tau_i'$
20     $\epsilon' \leftarrow \frac{1}{2}{\cdot}\epsilon'$
21   return $S, \overline{\phi}$
```

The algorithm follows the refinement approach sketched in Section V-B. It starts by sampling the black box at the two parameter valuations $(\mathbf{1},\mathbf{0})$ and $(\mathbf{1},\boldsymbol{\mu}^{\max})$, where $\mathbf{1} = (1,\ldots,1), \mathbf{0} = (0,\ldots,0)$ in line 4. These two samples are sufficient to derive finite upper and lower bounds on the black box for *all* considered parameter valuations.

The outer loop starts with loose precision requirements $\boldsymbol{\tau}'$ and ϵ' (line 5-6), which are strengthened by a factor of two in each iteration (line 19 and line 20, respectively), until they reach $\boldsymbol{\tau}$ and ϵ. This implicitly results in a binary search for discontinuities in the black box.

For the given requirement on $\boldsymbol{\tau}'$, the inner loop successively samples the black box at the parameter valuations that maximize the difference between upper and lower bounds until they differ by at most ϵ: an AST ϕ_{diff} representing $UB_S(\boldsymbol{\lambda},\boldsymbol{\mu}+\frac{\boldsymbol{\tau}'}{2}) - LB_S(\boldsymbol{\lambda},\boldsymbol{\mu}-\frac{\boldsymbol{\tau}'}{2})$ is computed in line 13. This is done using the two operations *sub* and *trans*, which compute the difference between two ASTs and the translation of a given AST, respectively. In line 14, the operation *max* determines the parameter valuation $(\boldsymbol{\lambda}^{\delta},\boldsymbol{\mu}^{\delta})$ that maximizes ϕ_{diff} and the value δ that ϕ_{diff} assumes on this valuation. The implementations of *sub*, *trans*, and *max* are explained in the appendix.

By construction, if the algorithm terminates, $\overline{\phi}$ satisfies the precision requirements. It remains to argue why Algorithm 3 always terminates. Intuitively, this is because after sampling the black box at a particular parameter valuation $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ the algorithm will never have to sample the black box again in a neighborhood of $(\boldsymbol{\lambda}, \boldsymbol{\mu})$. This is shown in the following theorem:

Theorem 2 (Sampling Density, Lin. and Mon. Parameters): For every $\epsilon \in \mathbb{R}_{>0}$, every $\boldsymbol{\tau} \in \mathbb{R}^m_{>0}$ and every reasonable black box *BB*, there exists a $\delta \in \mathbb{R}_{>0}$, such that for all $\boldsymbol{\lambda}, \boldsymbol{\lambda}' \in \mathbb{R}^m_{\geq 0}, \boldsymbol{\mu}, \boldsymbol{\mu}' \in \mathbb{R}^n_{\geq 0}$:

$$UB_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}', \boldsymbol{\mu}'+\frac{\boldsymbol{\tau}}{2}) - LB_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}', \boldsymbol{\mu}'-\frac{\boldsymbol{\tau}}{2}) \leq \epsilon$$

$$\textit{if } \|\boldsymbol{\lambda}' - \boldsymbol{\lambda}\|_\infty \leq \delta \text{ and } \boldsymbol{\mu} - \frac{\boldsymbol{\tau}}{2} \leq \boldsymbol{\mu}' \leq \boldsymbol{\mu} + \frac{\boldsymbol{\tau}}{2}.$$

Here, $\|\boldsymbol{\lambda}\|_\infty$ is the maximum norm, i.e. $\|\boldsymbol{\lambda}\|_\infty = max\{|\lambda_1|, \ldots, |\lambda_m|\}$. For space reasons, the proof of the theorem can be found in the appendix. Using Theorem 2, it is easy to show that Algorithm 3 terminates:

Theorem 3 (Total Correctness of Algorithm 3): Algorithm 3 terminates on all inputs and returns a set of samples S that satisfies Equation (15).

Proof: By Theorem 2, Algorithm 3 will not add samples that are closer than $(\delta, \frac{\tau}{2})$ to previous samples. As the parameters are bounded, it can only do so a finite number of times. Thus it must eventually terminate. Its outputs are guaranteed to be correct due to the exit condition within the inner loop. ■

VI. A Precision-Timed Platform and Its Black-Box WCET Analysis

A. *Precision-Timed ARM: A Predictable Microarchitecture*

The PTARM [5] is a realization of a precision-timed [9] machine, a microarchitecture designed for predictable and repeatable performance. PTARM implements a subset of the ARMv4 ISA. In contrast to conventional architectures that use complex pipelines and speculation techniques to improve performance, which lead to non-predictable and non-repeatable timing, PTARM improves performance through predictable and repeatable hardware techniques. These include a thread-interleaved pipeline, scratchpad memories instead of caches, and a novel predictable DRAM controller [10].

B. *A Parameterized Timing Model for the PTARM*

The PTARM presented in [5] is unparameterized, i.e., the latencies of all instructions are fixed. However, there are several natural candidates for parameters:

- The latency of arithmetic and branch instructions, $\lambda_{\text{arithmetic}}$, determined by the processor frequency.
- The latencies of loads and stores to scratchpad, λ_{SPM}, determined by the type and speed of memory employed.
- The sizes of the instruction and data scratchpads, $\mu_{\text{I-SPM-Size}}$ and $\mu_{\text{D-SPM-Size}}$, which determine whether a memory access reaches the scratchpad or the DRAM.
- The latencies of loads, λ_{DRAM}, and stores, $\lambda_{\text{DRAM-Store}}$, to DRAM, determined by the DRAM chip and the configuration of the memory controller.

With the above parameters, we have parameterized the latencies given in [5] to those found in Table I. The parameters $\mu_{\text{I-SPM-Size}}$ and $\mu_{\text{D-SPM-Size}}$ determine whether a memory access reaches the scratchpad or the DRAM. We have also parameterized the PTARM simulator accordingly.

TABLE I. Latencies of selected PTARM instructions in terms of linear parameters.

Instructions	**Latency**	
Data Processing	$\lambda_{\text{arithmetic}}$	
Branch	$\lambda_{\text{arithmetic}}$	
	SPM	DRAM
Load Register (*offset*)	λ_{SPM}	λ_{DRAM}
Load Register (*pre/post-indexed*)	$\lambda_{\text{arithmetic}} + \lambda_{\text{SPM}}$	$\lambda_{\text{arithmetic}} + \lambda_{\text{DRAM}}$
Store Register (*all*)	λ_{SPM}	$\lambda_{\text{DRAM-Store}}$
Load Multiple (*all*)	$N_{reg} \cdot \lambda_{\text{SPM}}$	$N_{reg} \cdot \lambda_{\text{DRAM}}$
Store Multiple (*all*)	$N_{reg} \cdot \lambda_{\text{SPM}}$	$N_{reg} \cdot \lambda_{\text{DRAM-Store}}$
N_{reg}: This is the number of registers in the register list.		

C. *Black-Box WCET Analysis for the PTARM*

We have adapted OTAWA [11], the open toolbox for adaptive WCET analysis, to the parameterized PTARM described above. We chose OTAWA as it provides an ARM analysis frontend. Our approach is similar to the one taken by Banerjee [12] in his WCET analysis for the PTARM. OTAWA follows the de facto standard approach to separate timing analysis into a low-level analysis, which determines bounds on the execution times of basic blocks, and a path analysis, based on integer linear programming, that combines constraints on the possible flow of control (such as loop bounds) and the basic-block bounds provided by the low-level analysis to obtain a bound on the WCET of the program as a whole.

The execution time of all instructions within the PTARM is *independent* of the execution history. This renders non-parametric WCET analysis for the PTARM comparatively easy: The execution time of each basic block is determined by iterating over its instructions and summing up their individual execution times. Upon memory accesses and instruction fetches, depending on the accessed memory address, we need to account for either the scratchpad or the DRAM latency. For instruction accesses, this is easy, as the accessed address is available. For data accesses, we rely on a very simple address analysis, which is often imprecise. In case of uncertainty, we conservatively account for a DRAM access.

A major difference between our work and Banerjee's is that latencies of arithmetic, branch, and memory instructions, as well as memory sizes are controlled by parameters.As these parameters may take arbitrary rational values, we cannot always represent basic-block times as natural numbers. Thus, we have replaced fixed-size integers by arbitrary-precision rational numbers from the GNU Multiple Precision Arithmetic Library (GMP) [13] throughout the entire OTAWA toolbox.

For the path analysis, OTAWA employs LP_SOLVE. We experienced arithmetic overflow errors as LP_SOLVE relies on fixed-size number representations. Thus, we replaced LP_SOLVE by a version of PipLib that is internally using the GMP library to avoid such problems.

VII. Experimental Evaluation

A. *Experimental Setup*

We evaluate APTA on a subset of the Mälardalen benchmarks [14] listed in Table II. We had to exclude benchmarks involving floating-point computations, divisions implemented in software, recursion, and complex switch statements due to Otawa limitations.

Loop bounds were annotated based on a manual inspection of the source code. All benchmarks were compiled using the

TABLE II. BRIEF SUMMARY OF THE BENCHMARKS.

Name	Size [byte]	Brief description
adpcm	26852	Adaptive pulse code modulation algorithm.
bs	4248	Binary search for the array of 15 integer elements.
bsort100	2779	Bubblesort program.
crc	5168	Cyclic redundancy check computation on 40 bytes of data.
fdct	8863	Fast Discrete Cosine Transform.
fibcall	3499	Simple iterative Fibonacci calculation, to calculate fib(30).
insertsort	3892	Insertion sort on a reversed array of size 10.
janne_complex	1564	Nested loop program.
jfdctint	16028	Discrete-cosine transformation on a 8x8 pixel block.
matmult	3737	Matrix multiplication of two 20x20 matrices.
ns	10436	Search in a multi-dimensional array.
nsichneu	11835	Simulate an extended Petri Net.
qsort-exam	4535	Non-recursive version of quick sort algorithm.
statemate	52618	Automatically generated code.

GNU ARM toolchain including GCC version 4.3.2. Time measurements were performed on an INTEL CORE I7 920 running at 2.67 GHz with 12 GB of RAM. We chose 64 KB as the maximal value for both the instruction and the data scratchpad memories. Linear parameters, modeling latencies, may take any rational value between 0 and 10.

B. Evaluation Results

Our first evaluation goal is to confirm that the black-box WCET analysis over-approximates the timing of the parameterized PTARM, and to evaluate its precision. To this end, we determine for each benchmark the ratio between the black-box WCET estimate and the execution time determined using the PTARM simulator in a single simulation run with the inputs that are provided with the MÄLARDALEN benchmarks. As the value analysis in the black box is very simple and thus bound to be imprecise, we perform this comparison with all linear parameters, including the DRAM latencies, set to 1. This eliminates the influence of the value analysis from the results. The results of this analysis are illustrated in Table III. For some benchmarks the black-box estimate is very close to the simulation result, yet for others the ratio is extremely large. This is due to imprecise loop bounds and other constraints on the control flow, and to the fact that the input exercised during simulation does not represent the worst-case input, e.g., in the sorting tasks.

Next, we evaluate how the number of black-box WCET samples affects the *precision* of the parametric analysis results on unsampled parameter vectors. To this end, we modify Algorithm 3 to report upper and lower bounds whenever a new sample has been taken. This yields two ASTs, $\overline{\phi}_{P,i}$ and $\underline{\phi}_{P,i}$, corresponding to the lower and upper bounds on the black box, for each benchmark P in the set of benchmarks $\mathcal{P}$ and number of samples i. Then, we sample the parameter space uniformly at random 100 times. For each of the randomly drawn parameter valuations $(\boldsymbol{\lambda}_j, \boldsymbol{\mu}_j)$, we evaluate the black box BB_P, and the upper and lower bounds $\overline{\phi}_{P,i}, \underline{\phi}_{P,i}$, and determine their ratios:

$$r^{\text{over}}_{P,i,j} := \frac{[\![\overline{\phi}_{P,i}]\!](\boldsymbol{\lambda}_j, \boldsymbol{\mu}_j)}{BB_P(\boldsymbol{\lambda}_j, \boldsymbol{\mu}_j)} \text{ and } r^{\text{under}}_{P,i,j} := \frac{[\![\underline{\phi}_{P,i}]\!](\boldsymbol{\lambda}_j, \boldsymbol{\mu}_j)}{BB_P(\boldsymbol{\lambda}_j, \boldsymbol{\mu}_j)}.$$

We summarize these ratios by taking their geometric means $r^{\text{over}}_i, r^{\text{under}}_i$ over all benchmarks found in Table III. In Figure 3, we depict r^{over}_i and r^{under}_i for i between two[2] and 26. The experiment was performed with a precision target of $(\epsilon = 1024, \boldsymbol{\tau} = (0,0))$[3], which ensures that an arbitrary number of samples can be taken. We observe a strong precision

[2]We start at two samples, because Algorithm 3 performs two samples before entering the refinement loop.

[3]Where $\boldsymbol{\tau}$ refers to the monotone parameters $\mu_{\text{I-SPM-Size}}$ and $\mu_{\text{D-SPM-Size}}$.

TABLE III. PRECISION OF THE BLACK-BOX WCET ANALYSIS.

Name	Black Box (cycles)	Simulator (cycles)	Ratio
adpcm	9989637	1598152	6.25
bs	318	279	1.14
bsort100	998109	8293	120.36
crc	248231	116995	2.12
fdct	11262	11069	1.02
fibcall	1140	1131	1.01
insertsort	4965	2949	1.68
janne_complex	4048	753	5.38
jfdctint	14016	13951	1.00
matmult	755274	745669	1.01
ns	42550	42549	1.00
nsichneu	32339	15551	2.08
qsort-exam	2132100	11125	191.65
statemate	108766	2809	38.72

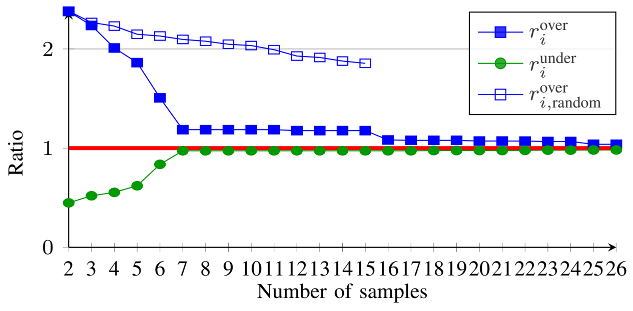

Fig. 3. Ratios between upper bound and black box $r^{\text{over}}_i, r^{\text{over}}_{i,\text{random}}$ and ratio between lower bound and black box r^{under}_i in terms of the number of samples i.

improvement on samples 3 to 7. After the first 7 samples, the obtained lower bound is very close to the actual black-box values for all benchmarks. The upper bound comes within 5% of the black box after 16 samples for most benchmarks, which is reflected by the geometric mean in the figure. For most benchmarks, the algorithm chooses to sample the black box at a different instruction scratchpad size than before, at samples 12 and 16, which yields a significant precision improvement.

As a baseline for Algorithm 3, we determine how well upper bounds based on a *random* set of samples approximate the black box. In Figure 3, $r^{\text{over}}_{i,\text{random}}$ denotes the ratio between these upper bounds, based on i random samples, and the black box. Random sampling yields much less precise estimates. In addition, computation times increase dramatically, which explains why we only perform this experiment for up to 15 samples. This demonstrates that some form of "intelligent" sampling is required for the black-box approach to be precise and efficient.

To evaluate analysis *efficiency*, we determined the analysis time of each benchmark up to and including the i^{th} sample. We decompose this analysis time into three components:

1) Invocations of the black box.
2) Operations on affine selection trees, e.g. minimization.
3) Invocations of PIPLIB.

In Figure 4 we show the geometric mean of the analysis time over all benchmarks up to the i^{th} sample. The bars are stacked, meaning that the top of the upper most bar reflects the overall analysis time. As the black box is called once for each sample, its contribution to the overall analysis time grows linearly. A superlinear growth is observed for the other two components, which is expected, as the problems to be solved grow with each sample. PIPLIB's contribution grows strongest, moving from the smallest to the largest share of analysis time. As observed in Figure 3, 16 samples usually result in very precise parametric upper bounds. On our benchmarks, 16 samples are processed in less than 2.2 seconds on the average.

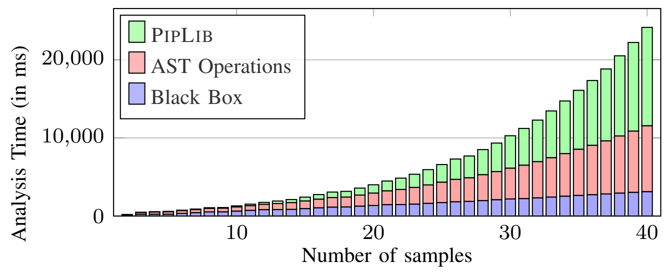

Fig. 4. Analysis time in terms of number of samples.

In Figure 3 we have shown how the *actual precision* depends on the number of samples. Next, we determine how many samples are required to meet a certain *precision guarantee*. To this end, we determine for each benchmark and for several precision targets $(\epsilon, (\tau, \tau))$[4] the number of samples $s_P(\epsilon, \tau)$ that Algorithm 3 takes until termination. We summarize these values in Figure 5. Three benchmarks, `nsichneu`, `adpcm`, `statemate`, ran out of memory for smaller values of τ. For the tightest precision requirement, $\epsilon = 32$, we report the maximum required number of samples $s_{32}^{\max}(\tau) = \max_{P \in \mathcal{P}} s_P(32, \tau)$ over all benchmarks for which the analysis terminated successfully, for τ between 8192 and 262144. For the loosest precision requirement, $\epsilon = 1024$, on the other hand, we report the minimum required number of samples $s_{1024}^{\min}(\tau)$ over all benchmarks. For ϵ between 32 and 1024, $s_P(\epsilon, \tau)$ is expected to lie between $s_{1024}^{\min}(\tau)$ and $s_{32}^{\max}(\tau)$. We also report the median number of samples $s_{256}^{\text{median}}(\tau)$ over all benchmarks for $\epsilon = 256$. For most benchmarks, the number of required samples is quite insensitive to ϵ: the median for $\epsilon = 256$ is close to the minimum for $\epsilon = 1024$.

VIII. Application to Commercial Microarchitectures

We have applied APTA to a parameterized version of the PTARM, a precision-timed architecture. It has been designed with the specific goal of reconciling performance and predictability. And indeed, as we demonstrate, precise and efficient architecture-parametric WCET analysis is feasible for the PTARM. Naturally, the question arises whether our approach is also applicable to other existing academic and commercial microarchitectures and whether a similar parameterization is reasonable for such microarchitectures.

To answer the second question first, we believe that the parameterization of the PTARM is quite typical for both commercial and research platforms: aspects that are often configurable in single-core processors are the processor frequency, the sizes of local memories (caches or scratchpads), and the interconnect, affecting memory access latencies, corresponding directly to the parameters in the PTARM. To use multi-core architectures in a hard real-time context, most of their shared resources will have to be partitioned (an approach promoted among others in the MERASA and Predator projects [15], [3]) in space and/or time: caches can be partitioned along their ways [16], buses and other interconnect can be partitioned in time by time division multiple access (TDMA) arbitration [17], and access to DRAM memory can be partitioned in time [18] and space [10]. Partitioning in space intuitively induces monotone parameters, whereas time-based partitioning may sometimes be modeled with linear parameters, however, caveats exist, as discussed below.

[4]Where (τ, τ) refers to the monotone parameters $\mu_{\text{I-SPM-Size}}$ and $\mu_{\text{D-SPM-Size}}$.

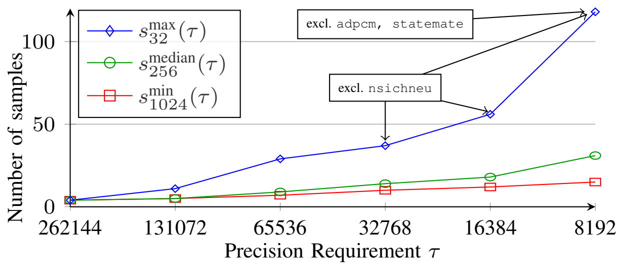

Fig. 5. Number of samples required to reach precision guarantee $(\epsilon, (\tau, \tau))$. Some benchmarks, namely `nsichneu`, `adpcm`, `statemate`, ran out of memory for $\epsilon = 32$ and are thus not included in $s_{32}^{\max}(\tau)$. The median and minimum, $s_{256}^{\text{median}}(\tau)$ and $s_{1024}^{\min}(\tau)$, could be determined for all values of τ.

For our approach to be applicable to a particular platform, its parameterized timing model needs to be linear or at least monotone in all of its parameters. This is, unfortunately, not the case for canonical models of many complex microarchitectures. In contrast to LRU, FIFO cache replacement is known to suffer from Belady's anomaly, i.e., under FIFO, increasing the cache's size may lead to a decrease in performance. For platforms including such non-monotone features, monotone parameterized timing models can be developed, however, at the cost of a loss in precision. Alternatively, if the goal is to find a system configuration statically, our analysis may also be performed on timing models that are *not* guaranteed to be monotone. This yields parametric WCET estimations that are not necessarily safe. Once a system configuration has been determined based on such an estimation, the black box can still be used to verify whether the timing constraints can be met. If not, the parametric WCET estimation can be refined accordingly and the process would have to be iterated.

In addition to monotonicity, our approach requires timing to be decomposed into contributions that can be attributed to different components. Such a decomposition is natural for so-called *fully timing-compositional* architectures [3], [19]. An example of a fully timing-compositional commercial architecture is the ARM7 [3]. For more complex architectures such as the Infineon TriCore or the PowerPC 755, it is possible to create conservative compositional models. The resulting loss in precision may, however, be substantial and depends on the particular architecture and decomposition [19], and is a topic of ongoing research.

IX. Related Work

The work most related to ours is that of Seth et al. [20] on *frequency-aware static timing analysis*. They consider a scenario in which a processor supports dynamic frequency/voltage scaling (DVS). Prior, naive approaches to exploit DVS assumed that the execution time would increase by a factor of c if the frequency was scaled down by a factor of c. Seth et al. observe that this is usually extremely pessimistic as scaling down the core frequency does not affect memory latencies, which can make up a large portion of a program's execution time. To remove this pessimism, they sketch an approach to derive a formula that captures the WCET of a program as a linear function of the processor's frequency. They also show how such a formula can be used to derive the optimal frequency to meet all deadlines under earliest-deadline first (EDF) scheduling. The setting considered by Seth et al. [20] can be seen as a special case of the setting considered in this work, in which there is exactly *one* linear parameter (the processor frequency) and *no* monotone parameters.

There is a bulk of work on a topic termed *parametric timing analysis* [21], [22], [23], [24], [25], [26], [27]. The goal of this work is to determine how the WCET of a program depends on its *inputs*. As an example, the number of loop iterations of a matrix-multiplication routine depends on the sizes of the matrices that are being multiplied. This is orthogonal to the present work in which the dependence of the execution time on *architectural* parameters is analyzed.

Most of the approaches to "input-parametric" timing analysis parameterize the path analysis, sometimes by applying parametric integer linear programming rather than standard integer linear programming (ILP). Here, we illustrate why such approaches cannot be easily transferred to architecture-parametric timing analysis: the standard ILP formulation for path analysis, takes the following form:

$$\begin{aligned} WCET := \max \quad & \sum_{b \in \textit{Basic Blocks}} c_b \cdot f_b, \\ \text{s.t.} \quad & \textit{Control-flow constraints}, \\ & \forall b \in \textit{Basic Blocks} : f_b \geq 0 \end{aligned} \tag{19}$$

where c_b is a constant denoting a bound on the execution time of basic block b, and f_b is a variable encoding the frequency of executing basic block b. In "input-parametric" timing analysis, parameters are introduced in the control-flow constraints, e.g., one might add the constraint $f_b = 2 \cdot p$, if f_b is the loop header of a loop that is executed two times the initial value of the input parameter p. This yields a parametric linear program. However, for architecture-parametric analysis, the basic-block bounds depend on the parameters: replacing the basic-block bounds by variables that depend on parameters immediately yields a quadratic optimization problem, as the basic-block bounds are multiplied with their respective execution frequencies. We are not aware of solvers for such parametric quadratic programs.

Work on retargetable WCET analyzers [28], [29], [30], [31] is also orthogonal to ours: its goal is to support retargeting a WCET analyzer to a new architecture, sometimes based on formal descriptions of a microarchitecture [31]. Retargetable WCET analyzers may serve as black boxes in our framework.

Our algorithms rely on *affine selection trees* (ASTs) as a data structure to represent piece-wise linear functions. We have implemented multiple operations on ASTs, such as *minimum*, *translation*, and *subtraction*, in a rather ad hoc fashion. In the future, we plan to develop specialized BDD-like data structures similar to LinAIGs [32] or LDDs [33] to more efficiently represent and manipulate parametric linear programming results. LinAIGs and LDDs are not directly applicable here as they represent predicates $\mathbb{R}^n \to \mathbb{B}$, rather than linear functions $\mathbb{R}^n \to \mathbb{R}$. While functions can be encoded as predicates, some operations, such as *subtraction*, are non-trivial to realize on the predicate representation.

X. Discussion

We have introduced a general framework for *architecture-parametric timing analysis* (APTA). To evaluate its viability we have instantiated it for a parameterized version of the PTARM, a precision-timed architecture. The results of our experimental analysis are promising, and demonstrate that precise and efficient APTA is indeed possible.

Opportunities for future work remain. Can the requirements for APTA, in particular monotonicity, be relaxed? Many questions arise when it comes to exploiting the results of APTA:

- How can APTA be integrated into a design-space exploration that identifies an architecture that meets all timing constraints while minimizing other aspects, such as cost or energy consumption?
- How can scheduling algorithms be adapted to exploit APTA results at run time to reduce energy consumption or increase the performance of non-critical tasks?

Acknowledgements

We would like to thank Edward Lee and Helmut Seidl for discussions on this work, and the anonymous reviewers for their helpful remarks. This work was supported in part by the Saarbrücken Graduate School of Computer Science, which receives funding from the DFG as part of the Excellence Initiative of the German Federal and State Governments, an Intel Early Career Faculty Award, and by the NSF, award #0720882 (CSR-EHS: PRET).

References

[1] C. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," *26th Digital Avionic Conference*, October 2007.

[2] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE TCAD*, vol. 28, no. 7, pp. 956–965, 2009.

[3] R. Wilhelm *et al.*, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *IEEE TCAD*, vol. 28, no. 7, pp. 966–978, 2009.

[4] D. N. Bui, E. A. Lee, I. Liu, H. D. Patel, and J. Reineke, "Temporal isolation on multiprocessing architectures," in *DAC*, 2011.

[5] I. Liu *et al.*, "A PRET microarchitecture implementation with repeatable timing and competitive performance," in *ICCD*, September 2012.

[6] P. Feautrier, "Parametric integer programming," *RAIRO Recherche Opérationnelle*, vol. 22, no. 3, pp. 243–268, 1988.

[7] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press, 1963.

[8] B. Dutertre and L. M. de Moura, "A fast linear-arithmetic solver for DPLL(T)," in *CAV*, 2006, pp. 81–94.

[9] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *DAC*, 2007, pp. 264–265.

[10] J. Reineke *et al.*, "PRET DRAM controller: Bank privatization for predictability and temporal isolation," in *CODES+ISSS*. ACM, 2011, pp. 99–108.

[11] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "OTAWA: an open toolbox for adaptive WCET analysis," in *SEUS*, 2010, pp. 35–46.

[12] S. Banerjee, "Timing analysis for the precision timed ARM processor," University of Kiel, Tech. Rep. 1212, September 2012.

[13] T. Granlund *et al.* The GNU multiple precision arithmetic library.

[14] J. Gustafsson *et al.*, "The Mälardalen WCET benchmarks - past, present and future," in *WCET*, July 2010.

[15] T. Ungerer *et al.*, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66–75, 2010.

[16] K. J. Nesbit, J. Laudon, and J. E. Smith, "Virtual private caches," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 57–68, Jun. 2007.

[17] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM TODAES*, vol. 14, no. 1, pp. 2:1–2:24, Jan. 2009.

[18] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: a predictable SDRAM memory controller," in *CODES+ISSS*, 2007, pp. 251–256.

[19] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis – definition and challenges," in *CRTS*, 2013.

[20] K. Seth *et al.*, "FAST: Frequency-aware static timing analysis," *ACM TECS*, vol. 5, no. 1, pp. 200–224, Feb. 2006.

[21] E. Vivancos, C. Healy, F. Mueller, and D. Whalley, "Parametric timing analysis," *SIGPLAN Not.*, vol. 36, no. 8, pp. 88–93, Aug. 2001.

[22] J. Coffman, C. Healy, F. Mueller, and D. Whalley, "Generalizing parametric timing analysis," in *LCTES*. ACM, 2007, pp. 152–154.

[23] S. Altmeyer, C. Humbert, B. Lisper, and R. Wilhelm, "Parametric timing analysis for complex architectures," in *RTCSA*, 2008, pp. 367–376.

[24] S. Mohan *et al.*, "Parametric timing analysis and its application to dynamic voltage scaling," *ACM TECS*, vol. 10, no. 2, pp. 25:1–25:34, Jan. 2011.

[25] S. Bygde, A. Ermedahl, and B. Lisper, "An efficient algorithm for parametric WCET calculation," *Journal of Systems Architecture*, vol. 57, no. 6, pp. 614 – 624, 2011.

[26] E. Althaus, S. Altmeyer, and R. Naujoks, "Precise and efficient parametric path analysis," in *LCTES*. ACM, 2011, pp. 141–150.

[27] B. Huber, D. Prokesch, and P. Puschner, "A formal framework for precise parametric WCET formulas," in *WCET*, 2012, pp. 91–102.

[28] M. G. Harmon, T. Baker, and D. B. Whalley, "A retargetable technique for predicting execution time of code segments," *Real-Time Systems*, vol. 7, no. 2, pp. 159–182, 1994.

[29] K. Chen, S. Malik, and D. I. August, "Retargetable static timing analysis for embedded software," in *ISSS*. ACM, 2001, pp. 39–44.

[30] A. Colin and I. Puaut, "A modular & retargetable framework for tree-based WCET analysis," in *ECRTS*. IEEE, 2001, pp. 37–44.

[31] X. Li, A. Roychoudhury, T. Mitra, P. Mishra, and X. Cheng, "A retargetable software timing analyzer using architecture description language," in *ASP-DAC*, Washington, DC, USA, 2007, pp. 396–401.

[32] W. Damm *et al.*, "Automatic verification of hybrid systems with large discrete state space," in *ATVA*, vol. 4218, 2006, pp. 276–291.

[33] S. Chaki, A. Gurfinkel, and O. Strichman, "Decision diagrams for linear arithmetic," in *FMCAD*, November 2009, pp. 53–60.

APPENDIX

A. Operations on Affine Selection Trees

a) Subtraction: Given two ASTs ϕ and ψ, the following recursive procedure computes an AST $\mathit{sub}(\phi,\psi)$, such that $[\![\mathit{sub}(\phi,\psi)]\!]\sigma = [\![\phi]\!]\sigma - [\![\psi]\!]\sigma$ for all valuations σ:

$$\begin{aligned}
\mathit{sub}((c\ ?\ \phi_1 : \phi_2),\psi) &= (c\ ?\ \mathit{sub}(\phi_1,\psi) : \mathit{sub}(\phi_2,\psi))\\
\mathit{sub}(\phi,(c\ ?\ \psi_1 : \psi_2)) &= (c\ ?\ \mathit{sub}(\phi,\psi_1) : \mathit{sub}(\phi,\psi_2))\\
\mathit{sub}(\infty,\infty) &= \text{undefined}\\
\mathit{sub}(l,\infty) &= -\infty\\
\mathit{sub}(\infty,l) &= \infty\\
\mathit{sub}(l,m) &= l-m
\end{aligned}$$

b) Maximization: Given an AST ϕ, the following recursive procedure determines the parameter valuation $(\boldsymbol{\lambda},\boldsymbol{\mu})$ that maximizes $[\![\phi]\!](\boldsymbol{\lambda},\boldsymbol{\mu})$ and the value that ϕ assumes on this valuation:

$$\begin{aligned}
\mathit{max}(\phi,\boldsymbol{\lambda}^{\max},\boldsymbol{\mu}^{\max}) &= \mathit{max}(\phi, \bigwedge_{x\in\{\boldsymbol{\lambda},\boldsymbol{\mu}\}} 0\le x\le x^{max})\\
\mathit{max}((l\ge 0\ ?\ \phi:\psi),p) &= \max(\mathit{max}(\phi,p\wedge l\ge 0),\\
&\qquad \mathit{max}(\phi,p\wedge l\le 0))\\
\mathit{max}(\infty,p) &= (\infty,\bot)\\
\mathit{max}(l,p) &= \begin{pmatrix} \begin{array}{l}\max\ l\\ \text{s.t. } p\end{array}, & \begin{array}{l}\arg\max\ l\\ \text{s.t. } p\end{array}\end{pmatrix}
\end{aligned}$$

The procedure collects all of the constraints on a path from the root of the tree to each leaf. At each leaf, a linear program then yields the maximum value and corresponding valuation of the AST. $\max$, defined below, is used at each inner node of the AST to select the larger of the two tuples coming from its subtrees:

$$\max((r_1,(\boldsymbol{\lambda}_1,\boldsymbol{\mu}_1)),(r_2,(\boldsymbol{\lambda}_2,\boldsymbol{\mu}_2))) = \begin{cases} (r_1,(\boldsymbol{\lambda}_1,\boldsymbol{\mu}_1)) & : \text{ if } r_1 > r_2 \vee \\ & \quad (r_1 = r_2 \wedge (\boldsymbol{\lambda}_1,\boldsymbol{\mu}_1)\le(\boldsymbol{\lambda}_2,\boldsymbol{\mu}_2)) \\ (r_2,(\boldsymbol{\lambda}_2,\boldsymbol{\mu}_2)) & : \text{ otherwise}\end{cases}$$

c) Translation: Given an AST ϕ, the following recursive procedure computes an AST $\mathit{trans}(\phi,\sigma')$, such that $[\![\mathit{trans}(\phi,\sigma')]\!]\sigma = [\![\phi]\!](\sigma+\sigma')$ for all valuations σ:

$$\begin{aligned}
\mathit{trans}((l\ge 0\ ?\ \phi:\psi),\sigma') &= (\mathit{trans}(l,\sigma')\ge 0\ ?\\
&\qquad \mathit{trans}(\phi,\sigma'),\mathit{trans}(\psi,\sigma'))\\
\mathit{trans}(l,\sigma') &= l + [\![l]\!]\sigma' - [\![l]\!]\mathbf{0}
\end{aligned}$$

B. Termination of Parametric Analysis

The following lemma and theorem show that the neighborhood of a parameter valuation that has been sampled does not have to be sampled again, which is crucial for proving termination of Algorithm 3.

Lemma 1 (Sampling Density, Linear Parameters): For every $\epsilon\in\mathbb{R}_{>0}$ and every reasonable black box BB, there exists a $\delta\in\mathbb{R}_{>0}$, such that for all $\boldsymbol{\lambda},\boldsymbol{\lambda}'\in\mathbb{R}^m_{\ge 0}, \boldsymbol{\mu}\in\mathbb{R}^n_{\ge 0}$:

$$\mathit{UB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}) - \mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}) \le \epsilon \quad \textit{if } \|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty \le \delta.$$

Proof: Let $\delta = \frac{\epsilon}{2\cdot \mathit{BB}(\mathbf{1},\mathbf{0})}$. By construction, we have $\boldsymbol{\lambda}' \le 1\cdot\boldsymbol{\lambda} + \|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty\cdot\mathbf{1}$. Thus, we can apply Inequation 12 to get

$$\mathit{UB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}) \le 1\cdot\mathit{BB}(\boldsymbol{\lambda},\boldsymbol{\mu}) + \|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty\cdot\mathit{BB}(\mathbf{1},\mathbf{0}).$$

Similarly, we get

$$\mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}) \ge 1\cdot\mathit{BB}(\boldsymbol{\lambda},\boldsymbol{\mu}) - \|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty\cdot\mathit{BB}(\mathbf{1},\mathbf{0}).$$

For smaller values of $\mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu})$, the sample $(\mathit{BB}(\boldsymbol{\lambda},\boldsymbol{\mu}),(\boldsymbol{\lambda},\boldsymbol{\mu}))$ would be redundant, contradicting the assumption that the black box is reasonable. Combining the two inequalities yields

$$\begin{aligned}
&\mathit{UB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}) - \mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu})\\
&\quad\le 2\cdot\|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty\cdot\mathit{BB}(\mathbf{1},\mathbf{0})\\
&\quad\le 2\cdot\delta\cdot\mathit{BB}(\mathbf{1},\mathbf{0}) = \epsilon. \qquad\blacksquare
\end{aligned}$$

Theorem 2 (Sampling Density, Lin. and Mono. Parameters): For every $\epsilon\in\mathbb{R}_{>0}$, every $\boldsymbol{\tau}\in\mathbb{R}^m_{>0}$ and every reasonable black box BB, there exists a $\delta\in\mathbb{R}_{>0}$, such that for all

$$\mathit{UB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}'+\frac{\boldsymbol{\tau}}{2}) - \mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}'-\frac{\boldsymbol{\tau}}{2}) \le \epsilon$$
$$\textit{if } \|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty \le \delta \text{ and } \boldsymbol{\mu}-\frac{\boldsymbol{\tau}}{2} \le \boldsymbol{\mu}' \le \boldsymbol{\mu}+\frac{\boldsymbol{\tau}}{2}.$$

Proof: Again, let $\delta = \frac{\epsilon}{2\cdot \mathit{BB}(\mathbf{1},\mathbf{0})}$. By monotonicity of *UB* (*) and *LB*, and by Lemma 1, we have:

$$\begin{aligned}
&\mathit{UB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}'+\tfrac{\boldsymbol{\tau}}{2}) - \mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}'-\tfrac{\boldsymbol{\tau}}{2})\\
\overset{(*)}{\le}\ &\mathit{UB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu}) - \mathit{LB}_{\{(\boldsymbol{\lambda},\boldsymbol{\mu}),(\mathbf{1},\mathbf{0})\}}(\boldsymbol{\lambda}',\boldsymbol{\mu})\\
\overset{\text{Lemma 1}}{\le}\ &2\cdot\|\boldsymbol{\lambda}'-\boldsymbol{\lambda}\|_\infty\cdot\mathit{BB}(\mathbf{1},\mathbf{0})\\
\le\ &2\cdot\delta\cdot\mathit{BB}(\mathbf{1},\mathbf{0}) = \epsilon \qquad\blacksquare
\end{aligned}$$