Appears in: IEEE Embedded Systems Letters, Volume: 11, Issue: 2, 2019

Combining Automated Measurement-Based Cost Modeling with Static Worst-Case Execution-Time and Energy-Consumption Analyses

Volkmar Sieh, Robert Burlacu[1], Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat

Department of Computer Science, Distributed Systems and Operating Systems
[1]Department of Mathematics, Economics · Discrete Optimization · Mathematics

Friedrich-Alexander University Erlangen-Nürnberg (FAU)

***Abstract*—Predicting the temporal behavior of embedded real-time systems is a crucial but challenging task, as it is with the energetic behavior of energy-constrained systems, such as IoT devices. To carry out static analyses in order to determine the worst-case execution time (WCET) or the worst-case energy consumption (WCEC) of tasks, cost models are inevitable. However, these models are rarely available on a fine-grained level for commercial-off-the-shelf hardware platforms.**

In this paper, we present NEO, an end-to-end toolchain that automatically generates cost models, which are then integrated into an existing static-analysis tool. NEO exploits automatically generated benchmark programs, which are measured on the target platform and investigated in a virtual machine. Based on the gathered data, we formulate mathematical optimization problems that eventually yield both worst-case execution-time and energy-consumption cost models. In our evaluations with an embedded hardware platform (e.g., ARM Cortex-M0+), we show that the open-source toolchain is able to precisely bound programs' resources while achieving acceptable accuracy.

***Index Terms*—execution-time modeling, energy-consumption modeling, worst-case execution time (WCET), worst-case energy consumption (WCEC), static analysis**

I. Introduction

Time and energy are first-class resources in today's embedded computing systems. On the one hand, embedded real-time systems must meet deadlines and thereby guarantee timeliness. On the other hand, modern embedded systems, especially in the Internet of Things, also have energy-resource constraints that have to be fulfilled to guarantee minimum uptimes of battery-operated systems. To accurately account for these resource constraints at design time of the systems, detailed knowledge on the runtime behavior, respectively the energetic behavior, of the system's software is crucial. Commonly, system designers carry out extensive measurements to consider resource constraints at design time. However, this approach has the two major drawbacks of being labor-intensive and not being able to guarantee safe upper bounds on the resource consumption of time or energy. To solve these problems, static code-analysis techniques were created that yield safe upper bounds for both the worst-case execution time (WCET) [1] and the worst-case energy consumption (WCEC) [2].

The basic working principle of these static-code analyses is subdivided into two phases: The hardware-independent path analysis of the program determines possible execution paths and, for example, yields upper bounds of loops in the control flow. By considering all possible execution paths, this phase enables solutions that capture all inputs. To then determine an actual WCET or WCEC bound for an arbitrary program with the help of results from the path-analysis phase, the second analysis phase requires a cost model of the target system's temporal or energetic behavior. However, the availability of these models, required for static code analysis, is a major problem: They are rarely documented for commercial off-the-shelf platforms and even if documentation is provided, its trustworthiness can be doubted [3], [4], leading to unsound results. To fill this information gap, previous approaches determined these models by using source-code benchmarks or hand-written assembly sequences [5], [6] being time-consuming and labor-intensive work.

To solve this problem, we present NEO, a toolchain for cost-model determination that relies on automatically generated benchmark programs. Each generated benchmark is executed once on the specific target platform while the execution time and the energy consumption are being measured. Subsequently, each benchmark is simulated inside a virtual machine to determine the number of occurrences of each executed machine-code instruction. From the data that is gathered in the measurement and the simulation phases, we formulate mathematical optimization problems, whose solutions determine the instruction-level hardware-cost model for the target machine. Finally, we integrate these models into an existing static worst-case analysis tool [7] to enable predictions for the WCET and the WCEC for arbitrary programs, which need to be safely run on the target hardware platform.

This paper presents the NEO approach [8] and discusses visions how NEO can be applied to further platforms. In Section III, the NEO's core concept is outlined, which is evaluated in Section IV. The subsequent Section V contains our visions how to extend the NEO approach in the future, Section II reviews related work, and Section VI concludes.

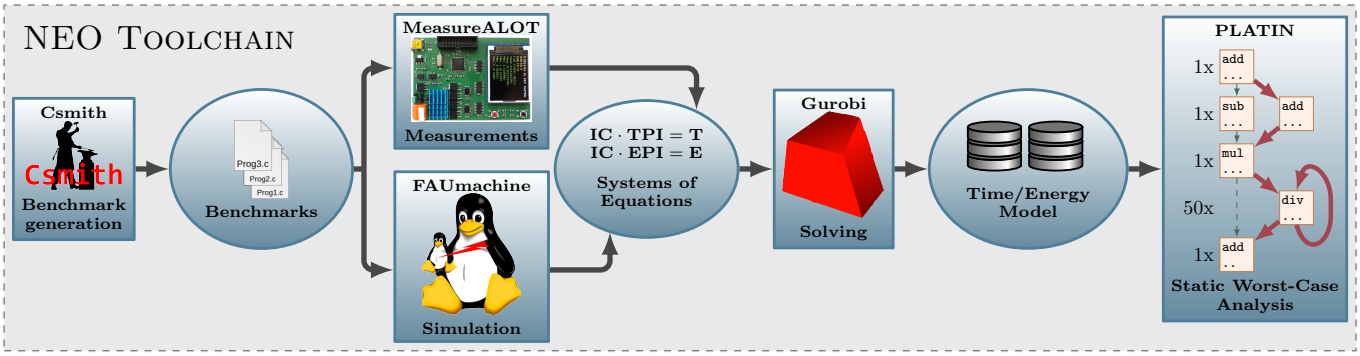

Fig. 1: NEO uses CSMITH to automatically generate benchmarks. The programs are executed on the target platform while time and energy are measured by the MEASUREALOT tool. The binaries are simulated on the virtual machine FAUMACHINE to determine the executed instructions. These counters and the measurements are used to formulate optimization problems, whose solutions determine the time & energy models, which are used in the analyzer PLATIN to determine WCET & WCET bounds.

II. Related Work

Previous research [5], [6], [9] that focuses on the creation of cost models (i.e., time and energy models) rely on *hand-crafted* benchmark suites or micro-benchmarks. In contrast, NEO does not depend on knowledge about such benchmarks and instead exploits *generated* benchmarks to automatically create time and energy cost models. Pioneering work by Tiwari et al. [5], for example, present an approach to extract an instruction-level power model for a RISC processor which requires manual measurements. Lee et al. [6] derive instruction-level power models with a regression analysis. The work, however, is limited to arithmetic operations on data. Pallister at al. [9] study the impact of instruction operand values on the energy demand. The authors use hand-crafted data to evaluate the energy demand of integer-based algorithms. On the contrary, NEO is not restricted to arithmetic operations, since the characterization of time and energy demand for branch instructions is inherently determined by the automated approach to benchmark generation. The automatic creation of benchmarks further releases the strict dependency on hand-crafted benchmarks as the programs, which are analyzed to construct the respective cost models. Sieh et al. [8] discuss further research related to NEO.

III. The NEO Toolchain

In this section, we first outline the theoretical, optimization-based approach (see Section III-A) and we subsequently present the practical implementation of the NEO toolchain (see Section III-B), which is summarized in Figure 1.

A. Optimization-Based Approach

For each kind of instruction i of a given instruction set I a value TPI_i (time per instruction i) must be measured giving the time the instruction needs for execution. Benchmark B's execution time T_B predictions are obtained by counting the instructions $IC_{B,i}$ to be executed and multiplying these numbers by the TPI model.

$$\sum_{i \in I} IC_{B,i} \cdot TPI_i = T_B$$

The core problem when determining hardware-cost models is that most machine-code instructions can have varying execution times and energy consumptions. For example, conditional branches take more CPU cycles, depending on if the branch is taken during runtime. Also, the caching behavior leads to varying execution times, when load instructions cause cache penalties. Consequently, each instruction i has a minimum ($TPI_{min,i}$) and a maximum ($TPI_{max,i}$) resource consumption, where the latter is used as models for our static worst-case analysis. Using different benchmark programs B we get an equation system of the form:

$$\sum_{i \in I} IC_{B,i} TPI_{min,i} \leq T_B$$
$$\sum_{i \in I} IC_{B,i} TPI_{max,i} \geq T_B$$

For energy consumption, we formulate the same system of equations with EPI instead of TPI and E_B on the equations' right sides, denoting the benchmarks' energy consumption.

NEO's core principle is to consider numerous benchmarks and thus equations. With this huge data pool, which is stored in matrices, we formulate mathematical optimization problems that eventually determine $TPI_{max,i}$ and $EPI_{max,i}$, describing the worst-case resource-consumption models of the target platform. These equation systems can be solved for $TPI_{max,i}$ and $EPI_{max,i}$ very fast using the Simplex Algorithm [10]. In our previous work [8], we describe how we use modified versions of the *least squares* and *least-sum-of-errors* approaches to compensate for measurement uncertainties and minimize errors in the cost models. NEO relies on automatically generated benchmarks to obtain such a data pool, which avoids using manually written benchmarks and thus time-consuming work. In the following, we describe how NEO determines instruction counts $IC_{B,i}$, execution times T_B, and energy consumptions E_B to determine TPI and EPI in our practical realization.

B. Practical Realization of the NEO Toolchain

In the current implementation of NEO, we use the following chain of tools, which is shown in the overview in Figure 1. In the first step, the CSMITH [11] tool is used to automatically generate a huge number of different C benchmark programs.

The toolchain uses GCC and CLANG to create executables from these benchmarks for the target hardware. Additionally, by employing different compiler-optimization levels, we obtain a higher diversity in the generated machine code in order to cover a wide range of execution scenarios. In Section V, we discuss techniques how to further improve the benchmarks' diversity, which is essential for the reliability of the produced cost model. The generated benchmarks are executed in the virtual machine FAUMACHINE [12] to gather the concrete number of instruction executions $IC_{B,i}$. All benchmarks are concretely executed once on the target platform while the execution time T_B and energy consumption E_B are being measured. We utilize the MEASUREALOT device [13] for both precise time and energy measurements.

Optimization problems are now formulated using the instruction counts IC_B, the measured execution times T_B, and energy consumptions E_B. These formulations focus on extracting the worst-case scenarios and eventually yield the instruction-level time and energy model. We use the GUROBI optimizer [14] to solve these problem formulations.

In the last step of the NEO toolchain, the automatically generated cost models TPI_{max} for execution time and EPI_{max} for energy consumption are integrated into the open-source, static worst-case analysis tool PLATIN [7]. From now on, PLATIN can be used to statically determine upper bounds for both the WCET and WCEC of arbitrary programs running on the target platform, for which the models were generated.

We point out that the used static worst-case analysis technique itself is proven to be sound [1]. However, due to the lack of documentation, our only resort is to rely on hardware measurements to determine the cost models, and this approach inherently can never be proven to be sound. Nevertheless, with our generated models for the ARM Cortex-M0+, we are able to determine upper bounds for all analyzed programs, as we demonstrate in the following evaluation section.

IV. EVALUATION

In this section, we summarize several evaluation results of NEO for the ARM Cortex-M0+ platform. With the limited complexity (i.e., few pipeline stages, small caches), it is representative for many embedded platforms. In the following, we first present evaluation results on static WCET analyses with the hardware-model determined by NEO. The capabilities of NEO with regards to creating energy-consumption models, which are the foundation for static analyses, and a comparison of NEO's model with an existing model are presented in [8].

To evaluate the validity of the entire NEO toolchain, we integrated the generated models into the existing WCET-analysis tool PLATIN [7]. With this setup, we measured the execution time of benchmarks from the TACLEBENCH suite [15] on our target platform and conducted a static WCET analysis of each benchmark. The results are summarized in Table 2. Depending on the structure and the complexity, we observed over-estimations ranging from 23 % (i.e., `binarysearch`) up to the pessimistic outliers up to 35,557 % (i.e., `fft`). The geometric mean of the normalized values of over-estimations is 208 % (i.e., an over-estimation factor of around 3). Considering the comparably small over-estimations due to the cost model [8], the pessimism in the results are due to conservative assumptions of the control flow within the path analysis, which can be improved by more advanced path analyses in the future. However, the main observation in this evaluation is that the toolchain determines bounds for all benchmarks, which demonstrates NEO's practical applicability.

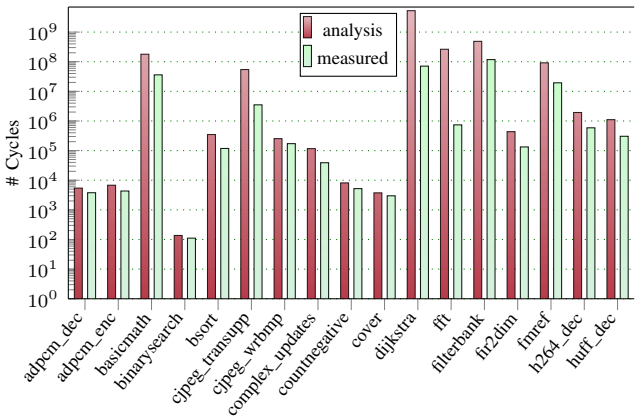

Fig. 2: The comparison (on logarithmic scale) of static WCET analysis (left bar) using the generated models with the execution trace (right bar) demonstrates that PLATIN is able to yield upper bounds for TACLEBENCH suite.

V. DISCUSSION & OUR VISION OF NEO

In this section, we discuss potential improvements within the toolchain's components, which we consider as future work.

A. Tailoring the Benchmark-Generation Process towards Micro-Architectural Awareness

For the NEO approach, it is essential that all CPU instructions appear somewhere in the benchmark programs. Otherwise, it is impossible to identify their impact on the systems' worst-case resource consumption. Consequently, we need benchmarks that, for example, use conditional-branch, integer-arithmetic, and floating-point instructions. If branch prediction influences the temporal/energetic behavior, NEO requires benchmarks that show poor branch-prediction scenarios. Similarly, benchmarks that lead to poor caching behavior are essential to reveal worst-case scenarios for load instructions.

Thus, we have to use a huge variety of benchmarks. The currently employed benchmark generator CSMITH creates many programs, but as it is targeted on finding bugs in compilers, it does not cover all scenarios presented above. NEO can, therefore, benefit from configurable benchmark generators, such as GENE [4], which can be extended to generate binaries that exhibit a high diversity in the processor's worst-case behavior. Since the tool works on a very low abstraction level (i.e., close to machine code), it offers the possibility to also generate memory-access sequences (i.e., code, data) that potentially trigger worst-case pipelining behavior. Additionally, tools exist to automatically determine the processor's cache-replacement policy [16], [17]. Their principle is also to generate tailored benchmarks and measure them subsequently on the target hardware platform while observing the temporal behavior. NEO can benefit from these approaches to gain awareness of micro-architectural effects, such as pipelining and caching.

B. Micro-Architecture Modeling by Machine Learning

The usage of machine-learning algorithms has been explored for the path-analysis problem of WCET analysis (i.e., finding loop bounds) [18]. With the increasing performance of machine-learning approaches, an interesting direction of cost modeling is the usage of neural networks to automatically learn feasible micro-architectural behavior (e.g., of the processor's pipeline). Our idea is feeding tailored benchmarks (i.e., different accesses to program and data memory from variable addresses), as described above, as input to the neural network where potential pipeline states are encoded as input for neurons. After training, the network then reveals feasible pipeline-state sequences being eventually useful for cost modeling.

C. Enhanced Measurement Support

We intend to enlarge the scope of NEO towards more-complex hardware architectures (e.g., x86). To accomplish that goal, different energy-measurement methods are required. As a generic measurement approach, we are working on employing current clamps together with the open-source oscilloscope[1].

Furthermore, many modern processors also offer integrated energy measurement facilities, such as Intel's Running Average Power Limit (RAPL) [19], where consumed energy is read via registers over a fixed duration. This approach relies on cost models integrated into the processor. Although this feature lacks a high temporal resolution, it significantly eases the setup of a benchmarking system. The NEO toolchain can be extended to make use of any of these measurement methods to address a broad range of hardware platforms.

D. Exploitation of Performance Counters

Together with the aforementioned energy-accounting registers, modern x86 processors also offer fine-grained performance counters [19]. These can be utilized to determine the number of cache misses/hits. Knowing these numbers improves the estimation of our cost models.

Regarding energy, we can split the energy needed by any `load` instruction into two parts. The first part modeling the energy consumption while fetching/decoding the instruction and accessing the first-level cache, which has to be considered in any case, and additionally, a second part modeling the energy needed to read from the second-level cache, which causes additional cost only in case of a first-level cache miss. Consequently, $IC_{ld} \cdot EPI_{ld}$ is replaced by $IC_{ld} \cdot EPI_{ld'} + MC \cdot EPMC$. The number of cache misses (MC) is obtained from the respective performance counter. $EPMC$ represents the amount of extra energy needed in case of a cache miss. This technique can be analogously applied to `store` instructions and can also be extended for more cache levels.

Furthermore, this approach can be used for branch instructions. Here, energy consumption is split into modeling branches with correct branch prediction and a penalty in case branch prediction fails. The number of mispredictions can be determined by reading performance counters. Taking branch prediction and caches into account will improve the generated energy and timing models on more-complex machines.

[1]Open-source oscilloscope RedPitaya: redpitaya.com

E. Robust Mathematical Optimizations

Measuring the amount of energy E_B needed while executing the benchmark with our MEASUREALOT device is prone to measuring uncertainties. We can handle this by adding a certain amount of energy to E_B for any benchmark program, which is typically too conservative. Instead, we pursue the Γ-robustness approach [20] from the field of robust optimization, where only a certain amount of benchmark problems are treated as uncertain, e.g., prone to measuring inexactness.

VI. CONCLUSION

NEO[2] automatically builds time and energy models for embedded systems without a priori knowledge, effectively analyses generated benchmarks, combines measurement-based time and energy analyses with existing WCET and WCEC tooling infrastructure, and uses mathematical optimization techniques. Future work will improve awareness of the micro-architectural behavior which enables handling of more complex platforms.

Acknowledgment. This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/9-2, no. SCHR 603/13-1, the CRC/TRR 89 (Project C1), the CRC/TRR 154 (Project B07), and the Bavarian Ministry of State for Economics under grant no. 0704/883 25.

REFERENCES

[1] P. Puschner and A. Schedl, "Computing maximum task execution times: A graph-based approach," *Real-Time Systems*, vol. 13, pp. 67–91, 1997.
[2] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the worst-case energy consumption of embedded software," in *Proc. of RTAS '06*, 2006.
[3] J. Abella et al., "WCET analysis methods: Pitfalls and challenges on their trustworthiness," in *Proc. of SIES '15*, 2015, pp. 1–10.
[4] P. Wägemann, T. Distler, C. Eichler, and W. Schröder-Preikschat, "Benchmark generation for timing analysis," in *Proc. of RTAS '17*, 2017.
[5] V. Tiwari and M. T.-C. Lee, "Power analysis of a 32-bit embedded microcontroller," *VLSI Design*, vol. 7, no. 3, 1998.
[6] S. Lee, A. Ermedahl, S. L. Min, and N. Chang, "An accurate instruction-level energy consumption model for embedded RISC processors," *SIGPLAN Notices*, vol. 36, no. 8, pp. 1–10, Aug. 2001.
[7] P. Puschner, D. Prokesch, B. Huber, J. Knoop, S. Hepp, and G. Gebhard, "The T-CREST approach of compiler and WCET-analysis integration," in *Proc. of SEUS '13*, 2013, pp. 33–40.
[8] V. Sieh, R. Burlacu, T. Hönig, H. Janker, P. Raffeck, P. Wägemann, and W. Schröder-Preikschat, "An end-to-end toolchain: From automated cost modeling to static WCET and WCEC analysis," in *Proc. of ISORC '17*, 2017, pp. 1–10.
[9] J. Pallister, S. Kerrison, J. Morse, and K. Eder, "Data dependent energy modelling: A worst case perspective," *CoRR, arXiv*, 2015.
[10] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.
[11] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and understanding bugs in C compilers," in *Proc. of PLDI '11*, 2011, pp. 283–294.
[12] M. Sand, S. Potyra, and V. Sieh, "Deterministic high-speed simulation of complex systems including fault-injection," in *Proc. of DSN '09*, 2009.
[13] T. Hönig, H. Janker, C. Eibel, O. Mihelic, R. Kapitza, and W. Schröder-Preikschat, "Proactive energy-aware programming with PEEK," in *Proc. of TRIOS '14*, 2014, pp. 1–14.
[14] Gurobi Optimization Inc., "Gurobi optimizer reference manual," 2016.
[15] H. Falk et al., "TACLeBench: A benchmark collection to support worst-case execution time research," in *Proc. of WCET '16*, 2016, pp. 1–10.
[16] T. John and R. Baumgartl, "Exact cache characterization by experimental parameter extraction," in *Proc. of RTNS '07*, 2007, pp. 65–74.
[17] A. Abel and J. Reineke, "Measurement-based modeling of the cache replacement policy," in *Proc. of RTAS '13*, 2013, pp. 65–74.
[18] M. Bartlett, I. Bate, and D. Kazakov, "Guaranteed loop bound identification from program traces for WCET," in *Proc. of RTAS '09*, 2009, pp. 287–294.
[19] Intel Corporation. (2013, June) Intel 64 and IA-32 Architectures Software Developer's Manual, Vol. 3B: System Programming Guide, Part 2.
[20] D. Bertsimas and M. Sim, "The price of robustness," *Operations Research*, vol. 52, no. 1, 2004.

[2]NEO is open source and available online: `gitlab.cs.fau.de/neo`