©2020 Published in 8th International Symposium on Innovative Technologies in Engineering and Science 23-25 October 2020 (ISITES2020 Bursa - Turkey)
https://doi.org/10.33793/acperpro.03.01.99

ABench2020: A WCET Benchmark Suite in Ada Programming Language

[1]Özge Gökçe and *[2]Veysel Harun Şahin
[1]Department of Computer and Information Engineering, Institute of Natural Sciences, Sakarya University, 54050, Sakarya, TURKEY
*[2]Department of Software Engineering, Faculty of Computer and Information Sciences, Sakarya University, 54050, Sakarya, TURKEY

Abstract

Validation is an important part of the development process of real-time systems. During validation, it should be proved that, the system meets its timing constraints. Therefore, worst-case execution time (WCET) analysis is performed. Currently, several WCET analysis tools are being developed. Researchers who develop new WCET analysis tools, need benchmarks to evaluate and compare their tools with alternatives. These benchmarks are called WCET benchmarks. In this paper a new WCET benchmark suite named ABench2020 is introduced. Its main focus is to provide benchmark programs in Ada programming language for WCET research. Therefore, ABench2020 includes several benchmark programs which were written in Ada programming language. The benchmark programs implement different program structures and properties to help researchers test their systems from different aspects. ABench2020 was published as open source. It is freely available over the Internet.

Key words: Real-Time Systems, Benchmark, Worst-Case Execution Time Analysis, Ada

1. Introduction

Benchmark programs and suites are widely used to compare computer systems, real-time systems, algorithms, methods, etc. Different kinds of benchmarks are used to evaluate different properties like execution time, energy usage, network performance and power characteristics. Benchmark applications are of critical importance to evaluate and compare newly developed methods, tools and systems.

Today, real-time systems [1], [2] are also widely used in several different areas such as industry, automotive, telecommunication, and Internet of Things. A real-time system is a type of computer system which is generally developed to perform some specific task(s) and works as part of a bigger system. Typically, real-time systems have little or no user interface, and they usually have power limits. In development of real-time systems, different programming languages are used like C, C++, Java [3], and Ada [4], [5].

One of the most important properties of real-time systems is time. Real-time systems have deadlines, which they need to meet during execution. Real-time programs should not miss their deadlines while producing correct result(s). Therefore, before deploying a real-time system, developers should measure and/or calculate the timing information of real-time program and make sure it meets its deadlines. Obtaining information about the timing behavior of the real-time

*Corresponding author: Address: Department of Software Engineering, Faculty of Computer and Information Sciences, Sakarya University, 54050, Sakarya, TURKEY. E-mail address: vsahin@sakarya.edu.tr. Phone: +902642955905

program is called worst-case execution time (WCET) analysis. Currently there are several tools and methods which help WCET analysis [6]–[9].

WCET analysis is an active research area, and new tools are developed currently. Researchers who develop new WCET analysis tools, need benchmark programs to evaluate and compare their work with alternatives. These benchmarks are called WCET benchmarks. The need of WCET benchmarks is the first motivation of this study. Benchmarks which help WCET analysis studies are generally developed using C programming language. However, WCET community also needs benchmark applications in different programming languages [10]. This is the second motivation of our study.

In this study, in accordance with our motivations we developed a new benchmark suite named ABench2020. It includes 12 benchmark programs written in Ada programming language. Each program in the benchmark suite implements a common algorithm in computer science. The contributions of this study are given below:

- We introduce a new benchmark suite named ABench2020, to help the evaluation and comparison of WCET analysis tools.
- It was developed using Ada programming language to meet the need of benchmark programs in different programming languages.
- It includes several programs which implement different program structures and properties to help the evaluation of WCET analysis tools from different aspects.
- It was published as open source, and freely available over the Internet [11].

The developed benchmark suite targets WCET community and aims to help advancing WCET research. Although it is developed for WCET community, it can also be used to assess the performance of different kinds of systems.

The paper is organized as follows. In Section 2, a literature review on benchmark is given. ABench2020 is explained in Section 3. Section 4 discusses the study and give information about possible future works.

2. Related Work

Benchmarking is an active study area on computers. Benchmarks are needed to measure, evaluate and compare different performance metrics of computer systems, tools and methods. There are numerous benchmarks developed for different purposes both in academia and industry. This section gives a summary of different benchmarks.

Several benchmarks on different areas are being developed by research community. For example, big data is a growing area which needs benchmarks. BigDataBench-S [12] is an open source benchmark suite developed to measure the performance of big data management systems. PARSEC benchmark suite [13] focuses on the evaluation of Chip-Multiprocessors (CMPs). It includes

parallel programs to help CMP research. Rodinia benchmark suite [14] targets heterogeneous computing. It aims the evaluation of multicore CPU and GPU platforms. Visual Road [15] focuses on the video database management systems (VDBMs). ompTGB [16] benchmark is based on OpenMP and aims to help the evaluation of real-time scheduling approaches on OpenMP based parallel tasks.

There are also both commercial and open source benchmarks which target the evaluation of computer systems from different perspectives. MiBench is a free benchmark suite for the evaluation of embedded systems [17]. Embedded Microprocessor Benchmark Consortium (EEMBC) [18] is an organization which provides benchmarks to asses different characteristics of embedded systems. Standard Performance Evaluation Corporation (SPEC) [19] develops different kinds of benchmark suites to evaluate various performance characteristics of computer systems.

Benchmarks for Java environments is another study area. For example, DaCapo benchmarks [20] target the evaluation of Java platforms. CD_x [21] focuses on the validation of real-time java virtual machines. JemBench [22] aims to help the evaluation of embedded Java platforms.

As mentioned above, this study focuses on benchmarks for WCET analysis works. Currently several benchmarks are available for the evaluation and comparison of WCET analysis tools. One of the WCET benchmark suites is the Mälardalen WCET Benchmarks [10]. It includes C programs and is mostly focused on flow analysis. PBench, is a parallel benchmark suite which aims to help the evaluation of WCET analysis tools from a multi-threaded perspective [23]. It is also open source and includes several sequential and parallel benchmark programs written in C programming language. MBBench [24] is another open source WCET benchmark suite. Its main focus is measurement-based WCET analysis tools. It is composed of C programs. TACLeBench [25] is an open source benchmark collection. It includes several benchmark programs from different research groups and tool vendors. The programs are in C programming language. PapaBench [26] and EMSBench [27] are two benchmarks which mimic real-time industrial applications. GenE [28] is a benchmark generator which generates benchmarks and provides flow facts. TASKers [29] generates real-time systems which helps the evaluation of WCET analysis and scheduling analysis techniques.

3. ABench2020

In this section we demonstrate ABench2020 in detail. In each part, a different aspect of the benchmark suite is explained.

3.1. Design and Development

The design of the benchmark was achieved in four stages: requirement analysis, operating system selection, feature selection, and algorithm selection.

In the **first stage** of the design of ABench2020, requirement analysis was performed. During requirement analysis, benchmark studies about WCET analysis were examined, and looked for

needed improvements. One of them was the need of benchmark programs in different programming languages other than C. According to this need, we decided to develop the benchmark suite in Ada programming language.

In the **second stage**, **operating system** support was determined. We chose to support Linux distributions, as they are open source, freely available, and widely used by research community. Although the benchmark programs were developed for Linux distributions, we believe that they can easily be compiled for other platforms which support Ada programming language as well.

The **third stage** is the **feature** selection stage. In this stage, programming structures and properties to be supported by the benchmark suite were determined. Feature selection is one of the most important parts of the design process. During the use of benchmark suite in evaluation, researchers want to test the behavior of their tools on different programming structures and properties. Therefore, benchmark suites should include several programs which implement different programming structures (decisions, loops etc.), and support different programming properties (single path, multi-threaded etc.). The feature list of ABench2020 is shown in Table 1.

Table 1. Feature List of ABench2020

Feature Name	Acronym	Description
Single Threaded	ST	Denotes that the program is sequential. It contains single thread of execution.
Multithreaded	MT	Denotes that the program is parallel. It contains multiple threads of execution.
External Routine	ER	Denotes that there is external routine call in the program.
Single Path	SP	Denotes that the program always follows the same path in each run. The flow of the program is always the same.
Multipath	MP	Denotes that the program may follow different paths in different runs. The flow of the program may change in different runs.
Dynamic Memory	DM	Denotes that the program makes dynamic memory allocation. It uses heap.
Loop	L	Denotes that the program includes loop structure.
Nested Loop	NL	Denotes that the program includes nested loop structure.
Decision	D	Denotes that the program includes decision structure.
Array	A	Denotes that the program uses array.
Recursion	R	Denotes that there is recursive function call in the program.
Bit Level Operation	BLOp	Denotes that bit level operation is present in the program like bit shifting etc.
Floating-Point Operation	FPOp	Denotes that the program performs operations on floating-point numbers.
Integer Operation	IntOp	Denotes that the program performs operations on integers.
Command Line Argument	CmdArg	Denotes that the program takes command line argument during startup.
File Input/Output	FileIO	Denotes that the program performs file input/output operations.

The **fourth stage** is the **algorithm** selection stage. In this stage, we determined the algorithms to implement in benchmark programs. The algorithms were chosen in a way that their implementation will require the usage of the features determined above. Therefore, a well-known algorithm of computer science for each program were selected.

After the design phase, the benchmark programs were developed using Ada programming language. The tests were performed on Pardus 19.3 Linux distribution. During tests, the programs

were compiled using GNAT (GNU Ada) 8.3.0.

3.2. Benchmark Details

There are 12 different benchmark programs in ABench2020. Each program implements a different algorithm and uses various features from Table 1. The programs and their supported features are shown in Table 2.

In the table, each row indicates a program in ABench2020, and each column indicates a feature. The acronyms of the feature names are written in the column headers. The plus (+) and minus (–) symbols in the rows indicate that the program supports the corresponding feature or not respectively. As can be seen from the table, the benchmark programs were selected specifically to cover most of the features in the feature list. Names of the programs were determined by considering the algorithms they implement. By doing this we aimed to help researchers to easily identify the operation of the program without looking into its code.

Table 2. Program List of ABench2020

Program Name	**Features**															
	ST	MT	ER	SP	MP	DM	L	NL	D	A	R	BLOp	FPOp	IntOp	CmdArg	FileIO
BinarySearchTree	+	–	–	+	–	+	+	–	+	+	+	+	–	–	–	–
BitwiseShift	+	–	+	+	–	–	–	–	–	–	–	+	–	–	–	–
Correlation	+	–	+	+	–	–	+	–	–	+	–	–	+	–	–	–
CorrelationFI	+	–	+	+	–	–	+	–	–	+	–	–	+	–	–	+
Factorial	+	–	+	+	–	–	+	–	–	–	–	–	–	+	+	–
Fibonacci	+	–	+	–	+	–	–	–	+	–	+	–	–	+	+	–
LinearSearch	+	–	–	+	–	–	+	–	+	+	–	–	–	–	–	–
LinkedList	+	–	–	+	–	+	+	–	+	+	–	–	–	–	–	–
MatrixMultiplication	+	–	–	+	–	–	+	+	–	+	–	–	–	+	–	–
PFactorial	–	+	+	–	+	–	+	–	+	–	–	–	–	+	+	–
SelectionSort	+	–	–	+	–	–	+	+	+	+	–	–	–	–	–	–
Softmax	+	–	–	+	–	–	+	–	–	+	–	–	+	–	–	–

BinarySearchTree program firstly creates a binary search tree and then traverses the newly created tree. The node values of the tree are hardcoded. Notable features of this program are dynamic memory allocation and recursive routine call. **BitwiseShift** program performs shift right and shift left operations on two hardcoded unsigned integers. This is the only program in ABench2020 which performs bit level operations. **Correlation** program calculates correlation of two samples. The samples are hardcoded. Two notable features are the loop structure and floating-point operation. **CorrelationFI** program is the replica of Correlation program with one difference. It performs file input operations. The samples are read from a text file. These two programs can also be used together to benchmark file I/O operations.

Factorial program takes input from command line and calculates the factorial of the input value.

Its notable features are loop structure and taking command line argument. Getting input from the command line makes it a good candidate for testing measurement-based tools. **PFactorial** program is the parallel version Factorial program. It includes all of the properties of Factorial program. Aside from these, its distinguishing feature is supporting multi-threaded operations. It calculates the factorial value by using two tasks (threads). **Fibonacci** program calculates Fibonacci value on a given index in the Fibonacci sequence. The index value is passed to the program as a command line argument. Notable features of this program are taking input from command line and recursive routine call.

LinearSearch program performs linear search operation on a hardcoded array of integers. Two notable properties are using dynamically allocated array and not using external routines. The index values of the search key are stored in a dynamically allocated array. It does not use external routine and hence self-contained. Therefore, it can be used for platform independent tests. **LinkedList** program creates a singly linked list. Its distinguishing feature is dynamic memory allocation. **MatrixMultiplication** program multiplies two matrices. It uses hardcoded matrices for calculation. Its notable feature is nested loop structure. It also does not use external routines. With these properties it can be suitable for platform independent tests on nested loop structures.

SelectionSort performs selection sort operation on a hardcoded array. Its two notable features are nested loop and decision structures. In addition, it also does not use external routines. It is useful for testing behaviors on decision structures which are embedded in nested loop structures. **Softmax** program implements softmax function. It performs calculation on a hardcoded array of integers. The results are floating-point numbers. Its notable features are loop structures and floating-point operations. It also does not use external routines. It is suitable for simple tests on loops.

3.3. Directory Organization and Contents

ABench2020 is stored in a repository over the Internet. Each program of the benchmark suite has its own directory. The name of the directory has the same name with the program. Each program directory contains 5 main files.

The **first file** is **source code** file of the program. This file has an “adb” file name extension which indicates that it is an Ada source code file. The **second file** is **Makefile** which manages the compilation process of the program. The **third file** is **call graph**. This is a directed graph which shows routine calls of the program. Call graph of the CorrelationFI program is shown in Figure 1. The nodes of the graph represent routines, and edges of the graph represent routine calls. The **fourth file** is **scope hierarchy graph**. This is a directed graph which shows routine calls and loop entries together. Scope hierarchy graph of the CorrelationFI program is shown in Figure 2. The nodes of the graph represent routines (e.g. CalculateCoefficient) and loops (e.g. CalculateCoefficient*L1). The edges represent routine calls and loop entries. Call graphs and scope hierarchy graphs help to understand the flow of the programs. The **fifth file** is **README** file which gives information about the related program. In addition to these files, there are also text files in the directories of some programs. These files can be either input or output file of the program. Explanations about these files are found in the README files of the related programs.

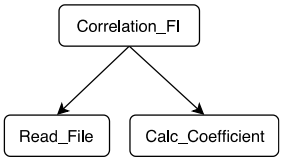

Figure 1. Call graph of the CorrelationFI program

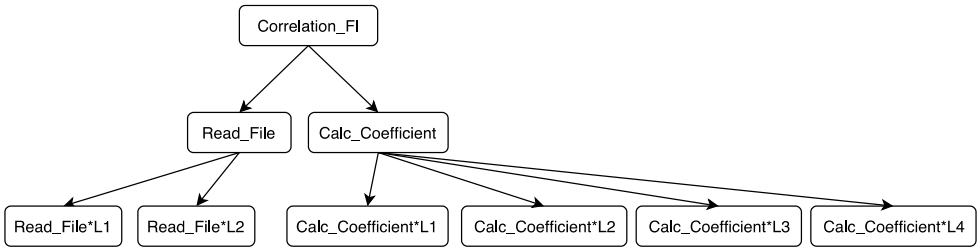

Figure 2. Scope hierarchy graph of the CorrelationFI program

4. Discussion and Future Work

ABench2020 specifically aims to help the evaluation and comparison of WCET analysis tools. Although its main focus is WCET analysis, it can also be used in the evaluation of real-time platforms and computer systems. Below we explain the similarities and differences between ABench2020 and other WCET benchmarks from different perspectives. In addition, we share our thoughts about possible future works on ABench2020 and WCET benchmarks.

4.1. External Routine

The programs in TACLeBench does not use external routines. One program in Mälardalen benchmarks uses external routines. The programs in MBBench and PBench use external routines. ABench2020 includes 6 programs which use external routines and 6 programs which does not use external routines.

4.2. Input

The inputs of the programs of TACLeBench and Mälardalen benchmarks are embedded in the source code. Linux versions of MBBench benchmarks take inputs either from command line or from file. Inputs of the RTEMS versions of MBBench benchmarks are embedded in source code. Two programs of PBench takes input from command line. In ABench2020, 3 programs get their inputs from command line and 1 program gets its input from file.

4.3. Parallelism

PBench is a parallel WCET benchmark suite. On the other hand, most of the programs of ABench2020 are sequential programs. Only one program (Factorial) has both sequential and

parallel versions. As multicore architectures become standard, more work on WCET analysis of parallel programs will be needed. Therefore, the need for parallel WCET benchmarks will increase.

4.4. Programming Language and Operating System

There are several WCET benchmarks in C programming language. ABench2020 was developed in Ada programming language to create an alternative to the research community. We believe that more WCET benchmarks in different programming languages are needed.

ABench2020 supports Linux distributions. There are currently many real-time operating systems (e.g. Zephyr, Apache Mynewt) are available and benchmarks which support them are needed. Therefore, in the future ABench2020 can be ported to different real-time operating systems.

4.5. Real-World Benchmarks

PapaBench and EMSBench mimic real-time industrial applications. AdaBench2020 does not include programs in this category. In the future, real-time industrial applications can be added and thus WCET benchmark ecosystem can be broadened.

4.6. Benchmark Development Procedure

Benchmark programs support different program structures and properties. By doing this, they enable the evaluation of systems from different perspectives. Therefore, feature lists are created during benchmark research. To help future research on WCET benchmarks, a standardized and comprehensive feature list can be prepared and presented to the WCET community. In addition, the steps taken to create benchmark suites from design to implementation can also be formalized and documented from a software engineering perspective. A guideline for benchmark studies can be provided to research community. This may also help future benchmark studies.

Conclusions

In this paper we introduced ABench2020 benchmark suite which has been developed to help the evaluation and comparison of WCET analysis tools. The benchmark suite was written in Ada programming language to meet the need of benchmark programs in different programming languages. It includes several programs which implement different program structures and properties to help the evaluation of systems from different aspects. It was published as open source, and freely available over the Internet [11]. We believe that ABench2020 will help WCET community to further research on WCET analysis. The quality of the WCET research can be greatly increased with a wealthy benchmark ecosystem.

Acknowledgements

We acknowledge that this work is supported by the Real-Time Systems Research Laboratory at

Sakarya University. ABench2020 can be accessed through the web page of the laboratory [11].

References

[1] Buttazzo GC. Hard Real-Time Computing Systems. Boston, MA: Springer US; 2011. doi: 10.1007/978-1-4614-0676-1

[2] Kopetz H. Real-Time Systems. 2nd ed. Boston, MA: Springer US; 2011. doi: 10.1007/978-1-4419-8237-7

[3] Bollella G et al. The Real-Time Specification for Java. Addison-Wesley; 2000.

[4] Dobbing B, Burns A. The Ravenscar Tasking Profile for High Integrity Real-Time Programs. Proceedings of the 1998 annual ACM SIGAda international conference on Ada - SIGAda '98; 1998; 1-6. doi: 10.1145/289524.289525

[5] AdaCore. AdaCore homepage. 20-Jul-2020. [Online]. Available: https://www.adacore.com

[6] Wilhelm R et al. The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst.; 2008; 7(3): 1–53. doi: 10.1145/1347375.1347389

[7] Davis RI, Cucu-Grosjean L. A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems. Leibniz Trans. Embed. Syst.; 2019; 6(1): 03:1-03:60. doi: 10.4230/LITES-v006-i001-a003

[8] Cazorla FJ, Kosmidis L, Mezzetti E, Hernandez C, Abella J, Vardanega T. Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey. ACM Comput. Surv.; 2019; 52(1): 14:1-14:35. doi: 10.1145/3301283

[9] Abella J et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. 10th IEEE International Symposium on Industrial Embedded Systems (SIES); 2015; 1–10. doi: 10.1109/SIES.2015.7185039

[10] Gustafsson J, Betts A, Ermedahl A, Lisper B. The Mälardalen WCET Benchmarks: Past, Present And Future. 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010); 2010; 136–146. doi: 10.4230/OASIcs.WCET.2010.136

[11] Sakarya University. Real-Time Systems Research Laboratory homepage. 20-Jul-2020. [Online]. Available: https://rtsrlab.sakarya.edu.tr

[12] Tian X et al. BigDataBench-S: An Open-Source Scientific Big Data Benchmark Suite. 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW); 2017; 1068–1077. doi: 10.1109/IPDPSW.2017.111

[13] Bienia C, Kumar S, Singh JP, Li K. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques - PACT'08; 2008; 72-81. doi: 10.1145/1454115.1454128

[14] Che S et al. Rodinia: A benchmark suite for heterogeneous computing. 2009 IEEE International Symposium on Workload Characterization (IISWC); 2009; 44–54. doi: 10.1109/IISWC.2009.5306797

[15] Haynes B, Mazumdar A, Balazinska M, Ceze L, Cheung A. Visual Road: A Video Data Management Benchmark. Proceedings of the 2019 International Conference on Management of Data - SIGMOD'19; 2019; 972–987. doi: 10.1145/3299869.3324955

[16] Wang Y et al. Benchmarking OpenMP programs for real-time scheduling. 2017 IEEE 23rd

International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA); 2017; 1–10. doi: 10.1109/RTCSA.2017.8046322

[17] Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB. MiBench: A free, commercially representative embedded benchmark suite. Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538); 2001; 3–14. doi: 10.1109/WWC.2001.990739

[18] EEMBC. Embedded Microprocessor Benchmark Consortium homepage. 20-Jul-2020. [Online]. Available: https://www.eembc.org

[19] Standard Performance Evaluation Corporation. Spec - Standard Performance Evaluation Corporation. 20-Jul-2020. [Online]. Available: https://www.spec.org

[20] Blackburn SM et al. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA'06; 2006; 169-190. doi: 10.1145/1167473.1167488

[21] Kalibera T, Parizek P, Haddad G, Leavens GT, Vitek J. Challenge Benchmarks for Verification of Real-Time Programs. Proceedings of the 4th ACM SIGPLAN Workshop on Programming Languages Meets Program Verification – PLPV'10; 2010; 57-62. doi: 10.1145/1707790.1707800

[22] Schoeberl M, Preusser TB, Uhrig S. The Embedded Java Benchmark Suite JemBench. Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems; 2010; 120–127. doi: 10.1145/1850771.1850789

[23] Serttaş S, Şahin VH. PBench: A Parallel, Real-Time Benchmark Suite. Academic Perspective Procedia; 2018; 1(1): 178–186. doi: 10.33793/acperpro.01.01.37

[24] Kuzhan M, Şahin VH. MBBench: A WCET Benchmark Suite. Sak. Univ. J. Comput. Inf. Sci.; 2020; 3(1): 39–49. doi: 10.35377/saucis.03.01.705777

[25] Falk H et al. TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research. 16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016); 2016; 55: 2:1–2:10. doi: 10.4230/OASIcs.WCET.2016.2

[26] Nemer F, Cassé H, Sainrat P, Bahsoun JP, Michiel MD. PapaBench: a Free Real-Time Benchmark. 6th International Workshop on Worst-Case Execution Time Analysis (WCET'06); 2006; 4: 1-6. doi: 10.4230/OASIcs.WCET.2006.678

[27] Kluge F, Rochange C, Ungerer T. EMSBench: Benchmark and Testbed for Reactive Real-Time Systems. Leibniz Trans. Embed. Syst.; 2017; 4(2): 02–1-02:23. doi: 10.4230/LITES-v004-i002-a002

[28] Wägemann P, Distler T, Hönig T, Sieh V, Schröder-Preikschat W. GenE: A Benchmark Generator for WCET Analysis. 15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015); 2015; 47: 33–43. doi: 10.4230/OASIcs.WCET.2015.33

[29] Eichler C, Distler T, Ulbrich P, Wägemann P, Schröder-Preikschat W. TASKers: A Whole-System Generator for Benchmarking Real-Time-System Analyses. 18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018); 2018; 63: 6:1-6:12. doi: 10.4230/OASIcs.WCET.2018.6