WCET TOOL CHALLENGE 2008: REPORT

Niklas Holsti[1], Jan Gustafsson[2], Guillem Bernat[3] (eds.),
Clément Ballabriga[4], Armelle Bonenfant[4], Roman Bourgade[4],
Hugues Cassé[4], Daniel Cordes[7], Albrecht Kadlec[5],
Raimund Kirner[5], Jens Knoop[6], Paul Lokuciejewski[7],
Nicholas Merriam[3], Marianne de Michiel[4], Adrian Prantl[6],
Bernhard Rieder[5], Christine Rochange[4], Pascal Sainrat[4],
Markus Schordan[6]

Abstract

Following the successful WCET Tool Challenge in 2006, the second event in this series was organized in 2008, again with support from the ARTIST2 Network of Excellence. The WCET Tool Challenge 2008 (WCC'08) provides benchmark programs and poses a number of "analysis problems" about the dynamic, run-time properties of these programs. The participants are challenged to solve these problems with their program-analysis tools. Two kinds of problems are defined: WCET problems, which ask for bounds on the execution time of chosen parts (sub-programs) of the benchmarks, under given constraints on input data; and flow-analysis problems, which ask for bounds on the number of times certain parts of the benchmark can be executed, again under some constraints. We describe the organization of WCC'08, the benchmark programs, the participating tools, and the general results, successes, and failures. Most participants found WCC'08 to be a useful test of their tools. Unlike the 2006 Challenge, the WCC'08 participants include several tools for the same target (ARM7, LPC2138), and tools that combine measurements and static analysis, as well as pure static-analysis tools.

1. Introduction

1.1 Worst-Case Execution-Time: Why It is Needed, How to Get It

The chief characteristic of (hard) real-time computing is the requirement to complete the computation within a given time or by a given deadline. The computation or execution time usually depends to some extent on the input data and other variable conditions. It is then important to find

1 Tidorum Ltd, Tiirasaarentie 32, FI 00200 Helsinki, Finland
2 School of Innovation, Design and Engineering, Mälardalen University, Box 883, S-721 23 Västerås, Sweden
3 Rapita Systems Ltd, IT Centre, York Science Park, York, YO10 5DG, United Kingdom
4 TRACES group, IRIT - Université Paul Sabatier, 118 Route de Narbonne, F-31062 TOULOUSE CEDEX 9, France
5 Institut für Technische Informatik, Treitlstraße 3/E182.1, Vienna University of Technology, A-1040 Wien, Austria
6 Institut für Computersprachen, Argentinierstraße 8/E185.1, Vienna University of Technology, A-1040 Wien, Austria
7 TU Dortmund University, Department of Computer Science XII (Embedded Systems Group), Otto-Hahn-Strasse 16, 44221 Dortmund, Germany

the *worst-case* execution time (WCET) and verify that it is short enough to meet the deadlines in all cases. For a multi-threaded computation the usual approach is to find the WCETs of each thread and then verify the system timing by some *scheduling analysis*, for example response-time analysis. Finding the WCET of each thread, and of other significant parts of the computation such as interrupt-disabled critical regions, is thus an important step in the verification of a real-time system.

Several methods and tools for WCET analysis have been developed. Some tools are commercially available. The recent survey by Wilhelm *et al*. [42] is a good introduction to these methods and tools. Some tools use pure static analysis of the program; other tools combine static analysis with dynamic measurements of the execution times of program parts. Unlike most applications of program analysis, WCET tools must analyse the *machine* code, not (only) the source code. This means that the analysis depends on the target processor, so a WCET tool typically comes in several versions, one for each supported target processor or even for each target system with a particular set of caches and memory interfaces. Some parts of the machine-code analysis may also depend on the compiler that generates the machine code. For example, the analysis of control-flow in switch-case statements may be sensitive to the compiler's idiomatic use of jumps via tables of addresses.

In general, WCET tools use simplifying approximations and so determine an *upper bound* on the WCET, not the *true* WCET. The pessimism, that is the difference between the true WCET and the upper bound, may be large in some cases. For most real, non-trivial programs a fully automatic WCET analysis is not (yet) possible which means that manual *annotations* or *assertions* are needed to define essential information such as loop iteration bounds. The need for such annotations, and the form in which the annotations are written, depends on both the WCET tool and on the target program to be analysed.

1.2 The WCET Tool Challenge: Aims and History

Several European developers and vendors of WCET tools collaborate in the Timing Analysis sub-cluster of the ARTIST2 Network of Excellence which is financed under Frame Programme 6 of the European Union [3]. Early in this collaboration, Reinhard Wilhelm (Saarland University) proposed a “competition” for WCET tools, modelled on the existing tool competitions, in particular tools for automated deduction and theorem proving (ATP).

However, discussion within the ARTIST2 group found several important differences between ATP tools and WCET tools. Where ATP tools have a standard input format (mathematical logic in textual form), WCET tools use different input formats: different source and machine languages for different instruction sets. Moreover, where ATP tools have a common and unambiguous expected result (a proof or disproof) and the true answer for each benchmark problem is known, WCET tools have a numeric, nearly continuous range of approximate outputs (WCET bounds or WCET estimates) and the true WCET of a benchmark program is often unknown, at least when the target system has caches or other dynamic accelerator mechanisms.

After some discussion, the term “Challenge” was chosen to emphasize that the aim is not to find a “winning” tool, but to challenge the participating tools with common benchmark problems and to enable cross-tool comparisons along several dimensions, including the degree of analysis automation (of control-flow analysis, in particular), the expressiveness and usability of the annotation mechanism, and the precision and safety of the computed WCET bounds. Through the

Challenge, tool developers can demonstrate what their tools can do, and potential users of these tools can compare the features of different tools.

As a result of this discussion, Jan Gustafsson of the Mälardalen Real-Time Centre organized the first WCET Tool Challenge in 2006 [13], using the Mälardalen benchmark collection [26] and the PapaBench benchmark [24], with participation from five tools. The results from WCC'06 were initially reported at the ISoLA 2006 conference [14] and later in complete form [15]. Lili Tan of the University of Duisburg-Essen did an independent evaluation of the tools on these benchmarks, also reported at ISoLA 2006 [36].

The second WCET Tool Challenge was organized in 2008 (WCC'08 [39]) and is the subject of this report. The report combines contributions from the WCC'08 participants and is edited by the WCC'08 steering group, some of whom are also WCC'08 participants. The editors wrote the introductory sections 1 and 2 (the participants contributing the tool descriptions in section 2.4) and the general discussions of problems and results in sections 3 and 4. The participants wrote section 5 to explain how they used their tools to analyse the WCC'08 benchmarks and to discuss their experiences and point out particular problems or successes. The editors wrote the summary and conclusion in section 6.

2. Organization of WCC'08

While WCC'06 was quite a success, it had some significant shortcomings. The main problem was the lack of a common target processor – all tools in WCC'06 assumed different target processors, and so produced incomparable WCET values for the same benchmarks. Another problem was the lack of test suites for the WCC'06 benchmark programs, which made it difficult for measurement-based or hybrid tools to participate – all WCC'06 results used static analysis only. Furthermore, many of these benchmarks were small or synthetic programs with a single execution path, and some were written in non-portable ways that made the analysis unduly difficult on some target processors. Finally, there was no precise definition of the Challenge "problems", that is, which parts of the benchmarks were to be analysed, and under which assumptions, and no common form for presenting the analysis results.

After WCC'06, a working group for the next Challenge was set up and consisted of Jan Gustafsson, Guillem Bernat, and Niklas Holsti. This became the steering group for WCC'08, with the aim of correcting the shortcomings of WCC'06: choosing a common target processor; choosing portable benchmark programs; creating test suites to let measurement-based tools participate; defining the analysis problems precisely enough to ensure that all participants make the same analyses; and defining a common result format for easy comparison. We were fortunate to get some financial support from ARTIST2 for this work.

The WCC'08 schedule was intended to produce results for presentation at the annual WCET Workshop, which in 2008 took place on July 1. However, the planning and structuring of the Challenge took some time to converge. The final version of the "debie1" benchmark, which became the main benchmark, was not available until April 9. Moreover, some participants had to port their tools to the ARM7 target which also caused delay. In the end, only one participant (OTAWA) produced results before the WCET Workshop; the rest entered their results later. For WCC'08 the

participants (tool developers) did all their own analyses. There was no “independent” analysis as done by Lili Tan for WCC'06 [36].

The main tool for organizing WCC'08 was the Wiki site hosted at Mälardalen [40]. We thank Hüseyn Aysan of MRTC for setting up the Wiki framework. Most of the initial Wiki content was written by Niklas Holsti with ARTIST2 funding. The Wiki defines the benchmark programs and the analysis problems – in other words, the questions that the participants should answer with their tools. For each benchmark program there is a page with result tables in which WCC'08 participants enter their results. WCC'08 participants have full editing access to the Wiki; non-participants have read-only access. In addition to the benchmarks, analysis problems, and result tables, the Wiki contains sundry information about the target processors and cross-compilers and a collection of questions and answers that arose during WCC'08.

2.1 The WCC'08 Benchmarks

For WCC'08 we wanted benchmark programs that are relatively large, preferably real programs, or based on real programs, rather than synthetic ones, and are provided with test suites for measurement-based WCET analysis. We also wanted new benchmarks to make a change from WCC'06. In the end, because of labour constraints, none of the WCC'06 benchmarks were included in WCC'08. We regret in particular the absence of PapaBench [24], caused by the lack of a test suite, and hope that it will be included in a future Challenge. WCC'08 defined five benchmarks: the “debie1” benchmark, courtesy of Space Systems Finland Ltd (SSF), and four benchmarks contributed by Rathijit Sen and Reinhard Wilhelm of Saarland University. The rest of this section briefly describes these benchmarks.

The “debie1” benchmark

The “debie1” benchmark is based on the on-board software of the DEBIE-1 satellite instrument for measuring impacts of small space debris and micro-meteoroids [11]. The software is written in C, originally for the 8051 processor architecture, specifically an 80C32 processor that is the core of the the Data Processing Unit (DPU) in DEBIE-1. The software consists of six tasks (threads). The main function is interrupt-driven: when an impact is recorded by a sensor unit, the interrupt handler starts a chain of actions that read the electrical and mechanical sensors, classify the impact according to certain quantitative criteria, and store the data in the SRAM memory. These actions have hard real-time deadlines that come from the electrical characteristics (hold time) of the sensors. Some of the actions are done in the interrupt handler, some in an ordinary task that is activated by a message from the interrupt handler. Two other interrupts drive communication tasks: telecommand reception and telemetry transmission. A periodic housekeeping task monitors the system by measuring voltages and temperatures and checking them against normal limits, and by other checks. The DEBIE-1 software and its WCET analysis with Bound-T were described at the DASIA'2000 conference [17]. The WCC'08 Wiki also has a more detailed description [40].

The real DEBIE-1 flight software was converted into the “debie1” benchmark by removing the proprietary real-time kernel and the low-level peripheral interface code and substituting a test harness that simulates some of those functions. Moreover, a suite of tests was created in the form of a test driver function. The benchmark program is single-threaded, not concurrent; the test driver simulates concurrency by invoking thread main functions in a specific order. The DEBIE-1

application functions, the test harness, and the test driver are linked into the same executable. This work was done at Tidorum Ltd by Niklas Holsti with ARTIST2 funding.

SSF provides the DEBIE-1 software for use as a WCET benchmark under specific Terms of Use that do not allow fully open distribution. Therefore, the “debie1” benchmark is not directly downloadable from the WCC'08 Wiki. Copies of the software can be requested from Tidorum[8]. SSF has authorized Tidorum to distribute the software for such purposes.

The benchmarks “rathijit_1” through “rathijit_4”

The goal of this group of benchmark programs, constructed and contributed by Rathijit Sen and Reinhard Wilhelm of Saarland University, is to stress-test both instruction-cache and data-cache analysers. The programs use a large number of C macros within loops. When these macros are instantiated by the C preprocessor the code size becomes quite large. The intent is to have a large instruction-cache footprint. The large code size means that these benchmarks also test the scalability of all analyses. There is no particular rationale or meaning behind choosing any particular constant, construct, condition, or ordering. They have been chosen at random. The programs are single-threaded.

The “rathijit_1” benchmark tests data reuse and control-flow analysis by initializing and walking through various parts of a 2D array. The “rathijit_2” benchmark aims to test control flow analysis, data-flow analysis, data reuse, and alias analysis. The basic unit has a 2-nested loop within a 3-nested conditional check and accesses some portion of 4 arrays through pointers which are set depending on the arguments passed to the basic unit. Globally there are 4 2D arrays, and the basic unit is instantiated a number of times. The goal of “rathijit_3” is to test code and data reuse in functions across different invocations. The program contains a 2-dimensional grid of functions, *func_i_j*, with *i* and *j* in 0 .. 10, for a total of 121 functions. The number of times each function is called depends on its position in the grid. Thus, *func_i_j* is called a number of times depending on *i* and *j* and should be able to reuse code and data fetched earlier, provided they have not been evicted from the cache. It is also possible that the data could have been fetched by some other function. The “rathijit_4” benchmark again tests control-flow analysis and data reuse by 4-nested switch-case statements, conditional branches and function calls from within the case statements.

The “rathijit” benchmarks are published under a liberal open-source licence and can be downloaded directly from the WCC'08 Wiki site [40].

2.2 The WCC'08 Analysis Problems

For each WCC'08 benchmark a number of *analysis problem*s or questions are defined, for the participants to analyse and answer. There are two kinds of problems: WCET-analysis problems and flow-analysis problems. Flow-analysis problems can be answered by tools that focus on flow-analysis (for example SWEET [35], unfortunately not a WCC'08 participant) but that do not have the “low-level” analysis for computing WCET bounds (for the ARM7 processor, or for any processor). Flow-analysis problems can also show differences in the flow-analyses of different WCET tools, and this may explain differences in the WCET bounds computed by the tools.

8 niklas.holsti@tidorum.fi

A typical WCET-analysis problem asks for bounds on the WCET of a specific subprogram within the benchmark program (including the execution of other subprograms called from this subprogram). For example, problem 4a-T1 for the "debie1" benchmark asks for the WCET of the *HandleTelecommand* function when the variable input data satisfy some specific constraints.

A typical flow-analysis problem asks for bounds on the number of times the benchmark program executes a certain statement, or a certain set of statements, within one execution of a root subprogram. For example, problem 4a-F1 for the "debie1" benchmark asks how many calls of the macro *SET_DATA_BYTE* can be executed within one execution of the function *HandleTelecommand*, under the same input-data constraints as in the WCET-analysis problem 4a-T1. By further requiring the analysis to assume that the execution time of *SET_DATA_BYTE* is arbitrarily large we make it possible for pure WCET-analysis tools to answer this flow-analysis question, since this assumption forces the worst-case path to include the maximum number of *SET_DATA_BYTE* calls; all alternative paths have a smaller execution time.

2.3 The WCC'08 Suggested Common Target Processor

After polling the potential participants, we decided to suggest the ARM7 processor as the common target for WCC'08. However, other targets were also allowed; the TuBound group used the C167. Since different ARM7 implementations may have different timing for memory accesses we picked a particular ARM7 chip, the LPC2138 from NXP Semiconductor [29]. The IF-DEV-LPC kit from iSYSTEM [18] was recommended for running LPC2138 benchmarks and was used by Tidorum and Rapita Systems. However, iSYSTEM no longer supply this kit in single quantities. The MTime group used another board, from OLIMEX [27]. The execution times should be the same on all LPC2138 boards because all timing interactions are on-chip and involve no off-chip components.

The ARM7 is basically a simple, deterministic processor that does not challenge the analysis of caches and complex pipelines that are important features of some WCET tools [42]. We had to choose such a simple common target because a more complex one would have required a large effort from most participants. Even so, some potential participants withdrew from WCC'08 because they did not have time to port their tools to the ARM7. More complex targets are under consideration for future Challenges and were certainly not excluded from the invitation to WCC'08.

The ARM7 architecture

The ARM7 [2] is a 32-bit pipelined RISC architecture with a single (von Neumann) address space. All basic ARM7 instructions are 32 bits long. Some ARM7 devices support the alternative THUMB instruction set, with 16-bit instructions, but this was not used in WCC'08. The ARM7 processor has 16 general registers of 32 bits. Register 15 is the Program Counter. Thus, when this register is used as a source operand it has a static value, and if it is a destination operand the instruction acts as a branch. Register 14 is designated as the "link register" to hold the return address when a subprogram call occurs. There are no specific call/return instructions; any instruction sequence that has the desired effect can be used. This makes it harder for static analysis to detect call points and return points in ARM7 machine code. The timing of ARM7 instructions is basically deterministic. Each instruction is documented as taking a certain number of "incremental" execution cycles of three kinds: "sequential" and "non-sequential" memory-access cycles and "internal" processor cycles. The actual duration of a memory-access cycle can depend on the memory subsystem. The

term “incremental” refers to the pipelining of instructions, but the pipeline is a simple linear one, and the total execution-time of an instruction sequence is generally the sum of the incremental times of the instructions

The LPC2138 chip and the MAM

The NXP LPC2138 implements the ARM7 architecture as a microcontroller with 512 KiB of on-chip flash memory starting at address zero and usually storing code, and 32 KiB of static on-chip random-access memory (SRAM) starting at adress 0x4000 0000 and usually storing variable data. There is no off-chip memory interface, only peripheral I/O (including, however, I2C, SPI, and SSP serial interfaces that can drive memory units).

The on-chip SRAM has a single-cycle (no-wait) access time at any clock frequency. The on-chip flash allows single-cycle access only up to 20 MHz clock frequency. At higher clock frequencies, up to the LPC2138 maximum of 60 MHz, the flash needs wait cycles. This can delay instruction fetching and other flash-data access, but the LPC2138 contains a device called the *Memory Acceleration Module* (MAM) that reduces this delay by a combination of caching and prefetching as follows.

The flash-memory interface width is 128 bits, or four 32-bit words. The MAM contains three 128-bit buffers that store, or cache, flash contents: the *Prefetch* buffer, the *Branch Trail* buffer, and the *Data* buffer. When the MAM is enabled, the Prefetch buffer holds the 128-bit block that contains the current instruction. The MAM concurrently prefetches the next 128-bit flash block into a fourth internal buffer called the “latch”. If execution continues sequentially from the last instruction in the Prefetch buffer, the next instructions are often already present in the MAM latch and are then moved to the Prefetch buffer so that execution continues without delay. The MAM again starts to prefetch the next 128-bit block from the flash to the latch. Instructions that read data from the flash make the MAM abort the prefetch, read the requested data from the flash while the processor waits, and restart the prefetch. This can force the processor to wait also for the next 128-bit instruction block. The MAM Data buffer caches the most recently read 128-bit block of flash data. This benefits sequential data access but is not useful for random access.

When a branch occurs, the processor must generally wait for the MAM to read the 128-bit block that contains the target instruction. The Branch Trail buffer in the MAM holds the last 128-bit block that has been the target of a branch. Thus, when an innermost loop has no internal branches, the loop-repeating branch usually “hits” in the Branch Trail buffer and causes no fetch delays.

The effect of the MAM on WCET analysis is similar to that of a cache. The state of the MAM depends on the dynamic history of the execution, and the state affects the time for instruction fetch and data access from the flash memory. The flash memory is divided into 128-bit blocks, similar to cache lines, and execution-timing can depend on which block an instruction or a datum lies in, and whether it lies in the flash or in the SRAM. For branches the timing can also depend on the offset of the source and target instructions within their blocks, through the block-prefetch function.

MAM configurations m2t1 and m2t3

In the available version of the LPC2138 the MAM has two bugs [25] that limit the useful MAM configurations to Mode 2 (MAM fully enabled). However, when the number of cycles for a flash access is set to 1 the MAM has no effect on timing, as if the MAM were disabled. We defined two MAM configurations for WCC'08: Mode 2 with 1 access cycle, and Mode 2 with 3 access cycles. These configurations are abbreviated *m2t1* and *m2t3*. In *m2t1* the MAM has no effect and timing is static. In *m2t3* the MAM has a dynamic effect on timing as discussed above.

Other factors that affect execution timing on the LPC2138

The on-chip peripherals in the LPC2138 connect to a VLSI Peripheral Bus (VPB) which connects to the Advanced High-performance Bus (AHB) through an AHB-VPB bridge. This bus hierarchy causes some delay when the ARM7 core accesses a peripheral register through the AHB. If the VPB is configured to run at a lower clock frequency than the ARM7 core this delay is variable because it depends on the phase of the VPB clock when the access occurs.

The programming tools

The IF-DEV-LPC kit from iSYSTEM comes with an integrated development environment called WinIDEA and a GNU cross-compiler and linker. The distributed benchmark binaries for WCC'08 were created with Build 118 of these tools using *gcc*-4.2.2 [19]. The IF-DEV-LPC kit has an USB connection to the controlling PC and internally uses JTAG to access the LPC2138. WinIDEA supports debugging with breakpoints, memory inspections, and so on.

The MTime group used another kit, the OLIMEX LPC-H2138 board [27] with *openocd* and *gcc*-4.2.1 as the development environment. This board has a serial I/O port which the MTime group found useful for software instrumentation techniques.

2.4 Tools Participating in WCC'08

The tools that participated in WCC'08 are (in alphabetical order) Bound-T, MTime, OTAWA, RapiTime, TuBound, and the WCET-aware C compiler wcc. Of these, only Bound-T and MTime also participated in the first Challenge in 2006. OTAWA, TuBound, and wcc are new tools and were not ready for use in 2006. RapiTime is a hybrid tool that needs a test suite for each benchmark, and such suites were not available for the WCC'06 benchmarks. Short descriptions of the WCC'08 participating tools follow, in alphabetical order by tool name.

Bound-T from Tidorum Ltd

Bound-T [37] is a typical static WCET analysis tool for simple processors. It loads an executable file, decodes instructions to build control-flow graphs and the call-graph, uses data-flow analysis to find (some) loop bounds, and IPET (per subprogram) to find WCET bounds. The data-flow analysis models the computation with Presburger arithmetic as explained in [16]. An assertion (annotation) language is provided. There is no cache analysis or other support for complex, dynamic processors. Bound-T is at present a closed-source tool that Tidorum licenses to users.

Bound-T is implemented in Ada. It uses no special formalisms to describe target processors or analysis algorithms, and no generative methods, but does use generic Ada modules. For example, a generic least-fixpoint solver is instantiated for various data domains and transfer functions to create a constant propagator, a def-use analyser, and other data-flow analyses.

An early version of Bound-T was used to analyse the original DEBIE-1 program [17] from which the WCC'08 benchmark “debie1” is derived. Moreover, the principal author of Bound-T (Holsti) took part in the creation of the DEBIE-1 program and also converted that program into the “debie1” benchmark for WCC'08 and defined the analysis problems for the benchmark. Thus Bound-T and Tidorum may have had some head start on the analysis of this benchmark for WCC'08.

MTime from Vienna University of Technology

MTime is a measurement based execution time analysis tool. It uses static analysis to split the program in smaller program segments (PS) with a manageable number of execution paths and uses model checking to generate test data [41]. The new version of the tool, which is currently under development, is also able to determine loop bounds by model checking [33]. Unfortunately, due to time constraints we were not able to prepare the new version for WCC'08. Thus we used the stable version of MTime within WCC'08, which is not able to analyze programs with loops.

The test data generated by MTime are used to enforce the execution of all feasible paths within each PS to perform the execution-time measurement. In the following calculation step the measured execution times are combined and an estimate for the WCET is calculated. Mutually exclusive paths within each PS are excluded but pessimism can be introduced by mutually exclusive paths spanning over more than a single PS. The obtained WCET estimate can be guaranteed on simple hardware without dynamic run-time optimization or run-time resource allocation, when all instructions have constant execution time and the CFGs of the source code and object code are the same.

The aim of the tool is to provide a convenient and platform-independent way to perform execution-time measurements. Since the tool operates on source code and uses a modular design, which makes it possible to use different instrumentation and measurement techniques by loading different modules, it is virtually platform independent.

OTAWA from the TRACES group at IRIT

OTAWA [8, 28] is a framework dedicated to the development of WCET analyzers. It includes a range of facilities like:

- loaders (to load the binary code to be analyzed – several ISAs are supported –, flow facts, a description of the hardware, *etc*.),
- code analyzers (*eg*. a CFG builder),
- code processors that perform specific analyses (*eg*. pipeline analysis, cache analysis, branch prediction analysis, etc.) and attach some properties to code items (*eg*. a time value can be attached to each basic block, a cache access category can be attached to an instruction, *etc*.)
- an IPET module that builds the IPET formulation from the CFG, the flow facts and the results of the invoked code processors.

OTAWA is an open software and has been designed to make the integration of new code processors easy. It is available under the LGPL licence.

RapiTime from Rapita Systems Ltd

RapiTime [32] is a measurement-based tool, *i.e.*, it derives timing information of how long a particular section of code (generally a basic block) takes to run from measurements. Timing information is captured on the running system by either a software instrumentation library, a lightweight software instrumentation with external hardware support, purely nonintrusive tracing mechanisms (like Nexus and ETM) or even traces from CPU simulators. The user has to provide test data from which measurements will be taken. Measurement results are combined according to the structure of the program to determine an estimate for the longest path through the program. The program structure is a tree that is derived from either the source code or from the direct analysis of executables. The user can add annotations in the code to guide how the instrumentation and analysis process will be performed, to bound the number of iterations of loops, *etc*.

RapiTime aims at medium to large real-time embedded systems on advanced processors. The tool does not rely on a model of the processor. Thus, in principle, it can model any processing unit (even with out-of-order execution, multiple execution units, various hierarchies of caches, *etc.*). Adapting the tool for new architectures requires porting the object code reader (if needed) and determining a tracing mechanism for that system. RapiTime is the commercial-quality version of the pWCET tool developed at the Real-Time Systems Research Group at the University of York [5].

TuBound from Vienna University of Technology

TuBound is a research prototype of a WCET-analysis and program-development tool-chain [30]. A distinctive feature of TuBound is that it allows to annotate programs with flow information at the source code level. This flow information is then transformed conjointly to code transformations within the development tool chain. TuBound currently includes a C++ source-to-source transformer, a static analysis component, a WCET-aware C compiler, and a static WCET analysis tool. The WCET analysis tool currently integrated into the TuBound tool-chain is *calc_wcet_167*, a static WCET analysis tool that supports the Infineon C167 as target processor.

WCET-aware C compiler wcc from Dortmund University of Technology

In contrast to other tools participating in this challenge, the *wcc* [12] is not a pure WCET analyzer but a complex compiler framework allowing an automatic WCET minimization. The WCET-aware compiler has been used for the development of different compiler optimizations driven by WCET information. Examples for WCET-aware optimizations are Procedure Cloning [23] or a cache-based Procedure Positioning [22].

The compiler consists of a high-level intermediate representation (IR), called *ICD-C IR*, and a low-level IR, called *LLIR*. This separation of the code representation offers potential for a wide range of different analyses and optimizations. The currently supported target hardware is the Infineon TriCore 1.3, a 32-bit microcontroller-DSP architecture optimized for high performance real-time embedded systems. The processor supports different memory hierarchies including caches, scratchpad memories and flashes.

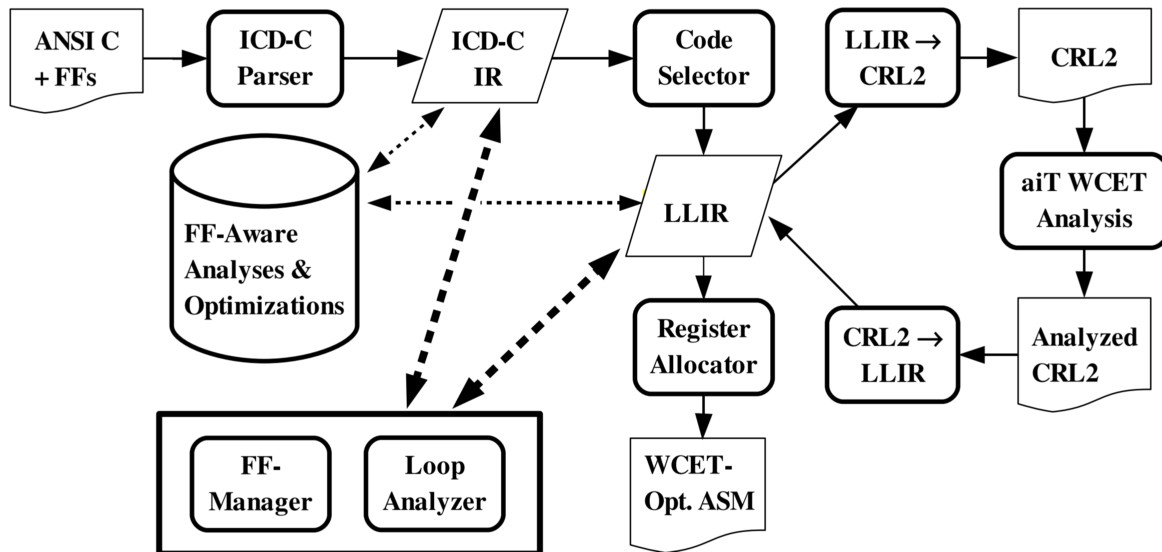

Figure 1: Workflow of wcc

The workflow is depicted in Figure 1. After parsing the C code, possibly annotated with flow facts (FF), into the ICD-C IR, standard compiler analyses, like data- and control-flow analyses, might be performed. In addition, the user can select from a large set of more than 30 average-case execution time (ACET) and WCET optimizations. The generation of the low-level IR is performed by the *code selector*. At the LLIR level, the user can choose again between standard analyses and numerous ACET and WCET optimizations. The code can be finally dumped into an assembly file and passed to the assembler.

Besides the typical components of an optimizing compiler, the *wcc* is extended by various WCET concepts. The fundamental extension distinguishing our compiler from other compilers is the binding of the compiler back-end with the static WCET analyzer *aiT* [1] developed by the company AbsInt. In a first step, the LLIR code is converted into CRL2, a machine-code intermediate representation, and used as input for the automatic invocation of aiT. After the WCET analysis, WCET and execution count information is imported back into the compiler and can be exploited for further applications, e.g. low-level WCET-driven optimizations.

The static loop analyzer [9] is another crucial module to turn a static WCET analysis framework into a fully automatic system. *wcc*'s loop analysis operates on the high-level IR and is based on Abstract Interpretation and polytope models. In addition, a technique called Program Slicing is deployed to accelerate the static analysis. The loop analyzer computes loop bounds and dumps them either in a human-readable format or passes them to the *Flow Fact (FF) Manager*.

The FF Manager is responsible for keeping flow facts, which are either read from the source code or generated by the loop analyzer, consistent. During optimizations, like Loop Unrolling, the original flow facts might become invalid. The manager keeps track of applied optimizations and automatically adjusts flow facts if required. In addition, the manager is responsible for the translation of ICD-C flow facts into the LLIR and further into CRL2. Thus, at any level of the code representation and after the application of optimizations, flow facts are correct.

To sum up, the *wcc* is a framework offering on the one hand standard compiler functionalities to translate an optimized C source code into machine code. On the other hand, the described extensions provide an automatic static WCET estimation of high- and low-level code for a state-of-the-art processor, compiler optimizations tailored towards an automatic WCET minimization and a static computation of flow facts. The latter is deployed to generate flow analysis results reported in the WCC'08.

3. Problems and Solutions

3.1 Life as Usual in the Embedded World

As usual for embedded programming, the participants that actually tried to run the benchmarks on a real processor had various problems with the development tools. For example, some versions and configurations of WinIDEA led to executables that did not initialize the global variables correctly; the bugs in the LPC2138 MAM [25] made some programs run incorrectly under some MAM configurations; and some participants experienced flash "lock-ups" that prevented reprogramming of the flash memory. Solutions to these problems were found by avoiding MAM modes 0 and 1 [25]; upgrading the development tools [19]; and using correct settings in obscure tool menus.

3.2 Problems in the Analysis

As these benchmarks are new ones, not analysed before by any WCC'08 participant, some problems were to be expected. Although the DEBIE program had been analysed for WCET earlier [17], that was for a different target – the Intel 8051, or 80C32 to be exact – not for the ARM7. In fact, for some participants this was their first extensive analysis of ARM7 code: the MTime and OTAWA groups ported their tools to the ARM7 target for WCC'08, and the ARM7 version of Bound-T, initially a mere prototype, was significantly extended for WCC'08.

Even so, most problems reported by the participants came not from the WCET tools, but from the wording of the definitions of the WCC'08 analysis problems or questions, some of which assume rather complex execution scenarios and constraints. The complexity was intentional, to stress the capability of the annotation languages. Indeed no tool was able to implement all constraints as flow-fact annotations. The obscurities and omissions in the definitions were not intentional.

4. Results

4.1 Tools and Targets

Table 1 below shows which target processors each participant has addressed for WCC'08 (most tools support other target processors, too). A notable fact is that four of six tools do flow analysis on the source-code level. This means that their flow-analysis results could in principle be compared in source-code terms. For example, on the source-code level we can talk about iteration bounds for specific loops, which is not possible on the machine-code level because of code optimizations. Future Challenges should perhaps pose flow question also on the source-code level. Another fact shown in Table 1 is that the suggested common target processor, the ARM7 LPC2138, is indeed

supported by many participants, at least in the simple *m2t1* mode. Note also that although *wcc* was used in WCC'08 only for source-code flow-analysis, it can produce WCETs for its target processors.

Table 1: Tools and targets in WCC'08

	Source-code flow analysis	***LPC2138 m2t1***	***LPC2138 m2t3***	***C167***
Bound-T		+		
MTime	+	+	+	
OTAWA	+	+	+	
RapiTime		+	+	
TuBound	+			+
wcc	+			

4.2 Results

The full set of results is too large to be presented here; please refer to the Wiki [40]. Table 2 below shows the number of analysis problems for each WCC'08 benchmark, the number of flow-analysis and WCET-analysis questions to be answered, and the number of questions answered by each participating tool (by 10 September 2008). If a tool answers the same question for several target processors, or for several MAM modes on the LPC2138, it still counts as only one answer. The absence of results, so far, for MTime and RapiTime is explained in section 5. Note that at present it is not certain that all participants have interpreted the analysis problems in exactly the same way (same input and execution constraints) which means that the current results (on the Wiki) from different tools may not be comparable even for the same target. We are working on this.

Table 2: Number of posed and answered analysis problems in WCC'08

Posed problems and questions										
Benchmark	***debie1***		***rathijit_1***		***rathijit_2***		***rathijit_3***		***rathijit_4***	
Number of problems	18		2		6		1		2	
Type of question	***Flow***	***WCET***	***Flow***	***WCET***	***Flow***	***WCET***	***Flow***	***WCET***	***Flow***	***WCET***
Number of questions	15	22	1	2	6	6	1	1	2	2
Answered questions (blank = none)										
Bound-T	13	18					1	1		
MTime										
OTAWA	7	16			2					
RapiTime										
TuBound	11	18		1		1		1		1
wcc	15		1		6		1		2	

WCC'08 actually has two kinds of analysis questions for WCET. The more common kind asks for the WCET of a given subprogram. The less common kind asks for the WCET of the interrupt-disabled regions within a given subprogram, which is an an important input to schedulability analysis. However, no participant answered questions of the second kind, although such problems have been studied before [7].

5. Tools and Experiences

This section collects the participants' reports on their goals, problems, solutions, and other comments on WCC'08. It is divided into a subsection for each participating tool, written by the developers of that tool, edited only for uniform formatting.

5.1 Bound-T (written by N. Holsti)

Adapting Bound-T to the ARM7

The basic ARM7 architecture is well suited to Bound-T. The only troublesome aspect is the lack of dedicated call and return instructions which makes it harder to split the target program into subprograms based on the machine code alone. For the ARM7 version of Bound-T, I took the timing of each ARM7 instruction from the “incremental” cycle numbers in the ARM7 manual [2]. The execution time of an instruction sequence is taken to be the sum of these numbers. In other words, I assumed that the incremental numbers include all possible pipeline stalls. I verified this timing model by measuring the execution time of suitable parts of the “debie1” benchmark on an IF-DEV-LPC kit. For these measurements, the software was instrumented with instructions to set the value of an LPC2138 output port (P0), the port was connected to a logic analyzer that recorded the value of the port on each clock, and the number of clock cycles between instrumentation points was computed from this record.

The value-analysis part of Bound-T at first had problems with the ARM7 code from *gcc*, because *gcc* often uses different registers as temporary pointers to stack variables. I extended the constant-propagation part of the value-analysis to handle semi-symbolic values of the form $P + c$, where P is a symbol representing the initial value of the stack pointer (the base of the stack frame for the subprogram under analysis) and c is a constant. References to such memory addresses are then resolved into references to statically identified parameters or local variables, even though the actual value of P is unknown. This extension will be useful for other targets, too, in particular when the compiler uses frame pointers for some, but not all subprograms.

Problems with the LPC2138

Bound-T has no general cache analysis, but I planned to model the LPC2138 MAM using a specific abstract interpretation. However, while the MAM at first seems rather simple, from the description in [29], with more study one quickly finds undocumented areas. For example, the detailed timing of aborting and restarting an instruction prefetch is not clear. To investigate these questions I made several test programs that systematically test many cases, for example branches with all 16 combinations of 128-bit-block offsets of the branch instruction and the target instruction. The test programs were densely instrumented with port outputs and measured with the logic analyzer. I found some interesting results, such as loops in which the execution time of the loop body alternates

between two values – faster on every other iteration, slower on the rest – even when every iteration executes the same instructions. I found no simple conceptual model that could explain the MAM timing. In desperation I tried to make a cycle-accurate simulation of the MAM, in the form of clocked registers and logic elements. This model accurately predicted most, but not quite all, of the observations. Running out of time and energy, I abandoned the MAM modelling, so for WCC'08 Bound-T/ARM7 can analyse only the *m2t1* configuration, where the MAM has no effect on timing.

Another, smaller problem with the LPC2138 is the location of the SRAM at the relatively large address of 0x4000 0000. The value-analysis in Bound-T depends on an external tool (the Omega Calculator [31]) that uses 32-bit integers and conservatively checks against overflow. However, this tool sometimes aborted the value-analysis of pointers into the SRAM because intermediate results threatened to overflow. I worked around this problem with annotations or by linking the benchmark programs with a memory map that uses a smaller SRAM address.

Problems and successes with the benchmarks

The Bound-T analysis of the “debie1” benchmark went in general as well as could be expected: many loop-bounds were found automatically, but many were not. Some of the WCC'08 execution constraints could be expressed in the Bound-T annotation language, but some could not, or required ugly work-arounds. For some “disjunctive” constraints I had to make several analyses with different annotations and combine the results manually. This is described in detail in the Bound-T notes on the WCC'08 Wiki site, including all annotations that I used. Of course, I had an unfair advantage over the other WCC'08 participants for this benchmark, because of my earlier association with the development of the DEBIE-1 software [17] and my work on the definition of the “debie1” benchmark.

Bound-T was much less successful with the “rathijit” benchmarks. The subprograms in “rathijit_1” and “rathijit_2” are too large for the value-analysis in Bound-T, which did not terminate in a reasonable time, or ran out of memory. Annotations can sometimes avoid the need for this value-analysis, but are no help in this case because Bound-T needs value-analysis to analyse the dynamically branching code from the numerous switch-case statements. Also, the large number of loops in these benchmarks would make manual annotation cumbersome. In “rathjiit_3” the compiled forms of the loops terminate by comparing the values of pointers to the LPC2138 SRAM area, and these values are too large for the value analysis in Bound-T. Relinking the program with a smaller SRAM address enabled successful automatic analysis of “rathijit_3”. In “rathijit_4”, the subprogram *func1* is compiled to an irreducible control-flow graph. The current version of Bound-T cannot analyze loops in irreducible graphs.

Comments on the WCET Tool Challenge

Considering WCC'08 as a participant, not as one of the organizers, I found it quite useful for driving the ARM7 version of Bound-T from a prototype towards a practical tool. The comparison of the Bound-T results to similar tools (in particular OTAWA) was a good check. The “rathijit” benchmarks will be good measures of future progress in the scalability of Bound-T. As an organizer, I enjoyed the contacts and discussions with all the participants.

5.2 MTime (written by B. Rieder and R. Kirner)

This section presents the results of the MTime tool for the WCET Tool Challenge 2008. Although it has not been possible to measure WCET results for the given benchmarks there are some results.

Target Hardware

First, a new instrumentation and measurement module for the old version of the MTime framework was created to support the ARM7TDMI platform. To perform execution-time measurements the OLIMEX ARM-USB-OCD Programmer and the LPC-H2138 development board were used [27]. The used target hardware provides an RS232 port on the development board and the programmer provides an ARM JTAG interface and an USB-to-RS232 converter on a single USB connector, which makes it the ideal choice for laptops or PCs without a serial port. An additional advantage is that the hardware is fully Linux compatible and fully supported by the *openocd* on-chip debugger while iSYSTEM, the manufacturer of the ITAG-U-ARM programmer and NXP LPC2138-M Mini-Target Board [18], neither supplies Linux software nor supplies information about the interface protocol of the ITAG-U-ARM JTAG programmer.

Measurements

The execution-time measurements are performed using the internal timer T0 of the LPC2138 which is set to run with the full CPU frequency. The measurement starts at the beginning of a PS with four assembly instructions which write “1” to the TCR0 register, which starts the timer, and ends at the end of the PS with another four instructions, writing a “0” to the TCR0 register for the timer to stop. The register width is 32 bits and there is an overflow register which is also 32 bits wide, resulting in a maximum counter value of 2^{64}, which should be sufficient for all applications. The current implementation uses a small boot loader, which is located in the flash, to download the application over the RS232 port. The test data and the measurement results are also transferred using the RS232 connection. The measured program resides in the SRAM. For this reason the MAM problem does not arise, but results are likely to be different from flash-based solutions.

WCET Tool Challenge Benchmarks

The stable version of the MTime framework does not support loops, therefore no benchmarks from the challenge could be performed.

Comments on the WCET Tool Challenge

The challenge is a very good opportunity to compare current execution time analysis frameworks. To increase the comparability of the individual tools, we propose to add more synthetic benchmarks, with small isolated problems. Benchmarks should be ordered by increasing complexity starting from single-path code and ending in applications with nested loops with data-dependent control flow. Adding complexity gradually would also increase the comparability of the results and point out individual weaknesses or strengths of individual analysis tools, providing valuable information for the both tool developers and users. Additionally, it would be interesting to measure not only the quality of the WCET bound but also the effort required for the preparation of the analysis (how much time to add annotations, *etc.*) and the time required for the analysis.

5.3 OTAWA (written by the TRACES group, see footnote 4 on title page)

General problems

The main problem we encountered is related to the specification of the "debie1" problems. First, we found it difficult to understand the meaning of some of the constraints and then we could not see how they should be translated into flow facts. We believe that defining a language to express the constraints would help, but this means that the constraints should probably be specified at a lower level (*eg*. line *x* in the *foo.c* source file – or instruction at address *a* in the executable code – is not executed, or is executed *n* times). A more formal description of the constraints could be more or less automatically taken into account by the tools. Second, many flow facts had to be determined manually (*eg*. loop bounds in the *memcpy* function according to the possible input values) and then specified manually to the WCET analyzer: this required much time for a single benchmark.

How we used OTAWA

To perform an automatic flow analysis, we used our oRANGE tool [10] which is not integrated to OTAWA at this time. oRANGE determines loop bounds from the source code. The flow facts related to other algorithmic structures as well as the loop bounds that could not be found automatically were specified as manual annotations.

Using the OTAWA framework, we have built a WCET analyzer that invokes the ARM binary code loader, the CFG builder, a flow fact loader that reads the flow facts provided by oRANGE as well as manual annotations, code processors that analyze the MAM (for instructions only) and the ARM7TDMI pipeline (they were specifically developed for the Challenge) and the IPET module. The MAM analysis is based on Abstract Interpretation [6] and the ARM7TDMI pipeline analysis uses execution graphs [34]. We reported results for the *m2t1* and *m2t3* MAM configuration. We considered that all the data were in the SRAM.

Comments on the WCET Tool Challenge

We found the Challenge very useful. First, it was one of our first experience with a "real" target (as part of our research activities, we are used to consider "generic" processor models). It also was the first time we really used our ARM loader. Second, the Challenge was an opportunity for people in the team to work for a common goal and to interact more deeply than usual.

5.4 RapiTime (written by G. Bernat and N. Merriam)

General comments

RapiTime observes and measures actual executions of the target program and therefore needs a suite of tests to be executed and observed. The "harness" module of the "debie1" benchmark contains a test suite, reached from the *main* function, that was created (by Tidorum Ltd) with two goals:

- To answer the analysis problems and questions defined for the "debie1" benchmark.

– To check that the modifications to the real DEBIE-1 flight software resulted in a benchmark program that still works as expected.

To satisfy the first goal, the test suite calls each "root" subprogram for each of the analysis problems defined for "debie1", under several different conditions. The calls are grouped according to the input and environment constraints for each analysis problem so that the measurements for each problem can be identified from the whole mass of measurements.

To satisfy the second goal, the test suite makes some checks on the state of the "debie1" application, after executing each test. These checks are not part of the measured code.

For some target systems, RapiTime can use non-intrusive methods for observing and measuring executions. However, for the present case an intrusive method based on instrumentation was used. The RapiTime C-code instrumenter inserted instrumentation points in the "debie1" source-code. During execution, the instrumentation points emit trace information (using LPC2138 port P0) which is time-stamped and recorded by external equipment. After execution, the RapiTime analyzer parses the trace, relative to the known control-flow structure of the program and the locations of the instrumentation points, computes the execution-time distribution of each measured block of code, and computes a WCET estimate using all possible execution paths.

The test-suite code contains special instrumentation points that delimit the traces relevant to each analysis problem defined for "debie1". Thus, while the benchmark program is only executed and measured once, the WCET analysis is done separately for each analysis problem. Moreover, RapiTime also computes coverage measures showing which parts of the code were executed.

The porting of RapiTime for the target was very simple, a single I/O port was used to trace the execution of the software using a logic analyser on an automatically instrumented version of the tests. The tests then were run and traces corresponding to each of the tests were collected and analysed successfuly.

We observed some rare but anomalously large execution times in the traces that indicate some transient behaviour of the processor. Only the *m2t3* configuration (MAM Mode 2, MAMTIM = 3) of the LPC2138 and "debie1" has been subjected to RapiTime analysis.

Even though RapiTime reports were generated, a lack of time to refine the analysis with annotations lead to unusable WCET results. The effort of completing the analysis and publishing the finished results will continue and will be reported in the next edition of the Challenge.

Comments on the WCET tool challenge

We believe that the Challenge is a very positive and that should be continued. This will strengthen the WCET community as a whole and unify the different research and development efforts.

5.5 TuBound (written by the TuBound group, see footnotes 5 and 6 on title page)

General remarks about the benchmarks and TuBound

At the current state of development, most effort went into the support of flow annotations and the transformation thereof. TuBound thus cannot yet cope with some of the additional constraints that were given in the problem descriptions.

This concerned mostly path annotations (*e.g.* function *f* will be executed at least once before the first invocation of function *g*) and variable value annotations, which caused overestimations in several benchmarks. For technical reasons, the interval analysis cannot yet accept user-supplied annotations. Work is underway to add support for this feature; however, it is not expected until after the deadline of the 2008 WCET Tool Challenge.

Although TuBound is conceived from ground up to be modular and to support multiple WCET analysis back ends, there was not enough time to port TuBound to the common ARM7 platform. We thus reported results for the C167 platform, which is a single-issue, in-order architecture with a jump cache.

About the WCET Tool Challenge

We found the WCET Tool Challenge a very good opportunity to evaluate the quality and performance of TuBound. It was a driver of many of TuBound's features to be implemented. The WCET Tool Challenge also revealed the necessity of supporting the proposed common target architecture to provide comparable results. We look forward to contributing to the next WCET Tool Challenge with a further enhanced version of TuBound. As a further improvement in comparability, we consider the availability of a highly expressive and widely supported common annotation language. Such a language has been demanded by our WCET'07 contribution on the “WCET Annotation Language Challenge” [20, 38], and a proposal on essential ingredients of such a language was contributed to the WCET'08 workshop [21]. A website has been created to collect contributions from the community to propose a common WCET annotation language [38].

5.6 Dortmund *wcc* (written by P. Lokuciejewski and D. Cordes)

Successes and suggestions for improvement

We considered only the flow-analysis problems, because we entered WCC'08 late (after the WCET 2008 workshop, in fact) and were pressed for time. Also, we saw no reason to enter WCET results for the current *wcc* target processor (TriCore) since no other WCC'08 participants use that target so no comparison would be possible.

The “debie1” benchmark is a complex real-world benchmark and its analysis is challenging for static analysis tools. However, its evaluation was a useful experience for us since it indicated which problems a static flow analysis must cope with in an industrial project. In the beginning, we had minor difficulties to figure out how some problems should be interpreted. However, with the intensive help of the organizers, all uncertainties could be removed and our static flow analysis succeeded in producing results for all flow analysis problems. To avoid interpretation problems for

the future challenges, we are of the opinion that a common flow-fact annotation language supported by all participants that enables a formal description of the constraints would be beneficial.

The second class of benchmarks, the "rathijit" benchmarks, could be all successfully analyzed. Due to the typical flow fact questions about function execution counts, no understanding problems occurred. In general, the WCC'08 showed that our loop analyzer is suitable for the flow analysis of complex software and if future WCET Tool Challenges feature our target processor, we would be challenged to compare our WCET analysis results with other participants.

Comments on the WCET Tool Challenge

We consider the Challenge as a valuable activity. It motivated us to extend our static loop analysis by further functionalities of practical relevance. In contrast to the originally provided loop bound information, our analyzer is now able to bound the number of calls to functions. The evaluation of the "debie1" benchmark also indicated some cases where our static loop analyzer originally produced unnecessary overapproximations. Driven by this fact, a thorough review of our code entailed some modifications that improved our tool's analysis precision. Last but not least, the Challenge emphasizes the needs for a formal flow-fact language to enable a comparison of different tools.

6. Conclusion

How can we evaluate the success of WCC'08? Compared to the 2006 Challenge, the number of participating tools grew from five to six, but of these six, four are new tools that did not take part in 2006, and three of the 2006 tools did not take part in 2008. So there was a major change in the participant set. One reason for this may be that the organizers gave too much emphasis to the suggested common target (LPC2138); some potential participants abstained from WCC'08 because they did not have time to create ARM7 versions of their tools. The participation of MTime and RapiTime means that WCC'08 met its goal of including measurement-based tools. We also succeeded in defining pure flow-analysis problems unrelated to WCET analysis, but this did not yet attract the participation of pure flow-analysis tools. Even *wcc*, the only participant that answered only flow-analysis problems (and answered all of them), is a WCET-oriented tool. An important goal for the next Challenge should be to motivate the 2006 participants to rejoin the Challenge, without losing the new 2008 participants.

The number of benchmark programs was smaller in 2008 (5 benchmarks) than it was in 2006 (17 benchmarks, counting the two programs in PapaBench as two benchmarks). However, the number of analysis problems, or questions, was increased, because each WCC'08 benchmark poses several questions. Perhaps the next Challenge should not make such large changes in the benchmark set and in the nature of the benchmarks. It should perhaps reintroduce some of the benchmarks from 2006 (PapaBench in particular [24]). The suggestion from the MTime team to add a sequence of small, synthetic benchmarks, posing analysis problems of increasing difficulty, is also attractive.

Most WCC'08 participants found it difficult to understand the constraints on input values and execution flows defined for some of the WCC'08 analysis problems, in particular for the "debie1" benchmark. It is striking that the reports from many participants in section 5 ask for a more formal

and standard way to define such things. This is certainly an issue for the organizers of the next Challenge, and also an opportunity to interact with the Annotation Language Challenge [20, 21].

Using a shared Wiki was successful. However, the next Challenge should strive to make the Wiki even more of a shared resource, jointly created and enjoyed by the Challenge participants, and indeed by any workers in the WCET analysis field.

With the conclusion of WCC'08, plans are being made for the next WCET Tool Challenge. The WCC'08 organizers suggest that the Challenge should be defined as a continuous process, allowing the addition of benchmarks, participants, and analysis results at any time, punctuated by an annual deadline. At the annual deadline, a snapshot of the results is taken and becomes the result of the Challenge for that year.

7. References

[1] AbsInt Angewandte Informatik GmbH, Worst-Case Execution Time Analyzer aiT for TriCore, 2008. http://www.absint.com/ait.

[2] Advanced RISC Machines, ARM7DMI Data Sheet. Document Number ARM DDI 0029E, Issue E, August 1995.

[3] ARTIST2 NoE – Cluster: Compilers and Timing Analysis, http://www.artist-embedded.org/artist/-Compilers-and-Timing-Analysis,45-.html.

[4] ARTIST2 Network of Excellence on Embedded Systems Design, http://www.artist-embedded.org/artist/.

[5] BERNAT, G., COLIN, A., and PETTERS, S.M., pWCET: a Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems, in: *Proceedings of the 3rd Int. Workshop on WCET Analysis* (WCET'2003), Porto, Portugal, 1 July 2003.

[6] BOURGADE, R., BALLABRIGA, C., CASSÉ, H., ROCHANGE, R., and SAINRAT, P., Accurate analysis of memory latencies for WCET estimation, in: *Int'l Conference on Real-Time and Network Systems* (RTNS), October 2008.

[7] CARLSSON, M., Worst Case Execution Time Analysis, Case Study on Interrupt Latency for the OSE Real-Time Operating System. Master's Thesis in Electrical Engineering, Royal Institute of Technology, Stockholm, Sweden, 2002-03-18.

[8] CASSÉ, H., ROCHANGE, C., University Booth at DATE 2007 (Design, Automation and Test in Europe), April 2007.

[9] CORDES, D., Loop Analysis for a WCET-optimizing Compiler Based on Abstract Interpretation and Polylib (in German). Master's Thesis, Dortmund University of Technology, 2008.

[10] DE MICHIEL, M., BONENFANT, A., CASSÉ, H., and SAINRAT, P., Static loop bound analysis of C programs based on flow analysis and abstract interpretation, in: *IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications* (RTCSA), August 2008.

[11] European Space Agency, DEBIE – First Standard Space Debris Monitoring Instrument, http://gate.etamax.de/edid/publicaccess/debie1.php.

[12] FALK, H., LOKUCIEJEWSKI, P., and THEILING, H., Design of a WCET-Aware C Compiler, in: *Proceedings of the 4th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia),* Seoul / Korea, October 2006.

[13] GUSTAFSSON, J., WCET Tool Challenge 2006, http://www.idt.mdh.se/personal/jgn/challenge/.

[14] GUSTAFSSON, J., The Worst Case Execution Time Tool Challenge 2006, in: S*econd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation* (ISoLA 2006), 2006.

[15] GUSTAFSSON, J., WCET Challenge 2006 – Technical Report, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-206/2007-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, http://www.mrtc.mdh.se/index.php?choice=publications&id=1209.

[16] HOLSTI, N., Computing Time as a Program Variable: A Way Around Infeasible Paths, in: *8th International Workshop on Worst-Case Execution Time Analysis* (WCET'2008), Prague, Czech Republic, July 1, 2008.

[17] HOLSTI, N., LÅNGBACKA, T., and SAARINEN, S., Using a Worst-Case Execution Time Tool for Real-Time Verification of the DEBIE Software, in: *Data Systems in Aerospace 2000* (DASIA 2000), EuroSpace and the European Space Agency, ESA SP-457.

[18] iSYSTEM AG, iF-DEV – iSYSTEM Free Tools for ARM7/ARM9/XScale, http://www.isystem.com/modules/news/article.php?storyid=25&location_id=0.

[19] iSYSTEM AG, Build 118 of iFDEV, http://www.isystem.si/SWUpdates/Setup_IFDEV_9_7_118/iFDEVSetup.exe.

[20] KIRNER, R., KNOOP, J., PRANTL, A., SCHORDAN, M. and WENZEL, I., WCET Analysis: The Annotation Language Challenge, in: *Post-Workshop Proceedings of the 7th International Workshop on Worst-Case Execution Time Analysis* (WCET 2007), (Pisa, Italy, July 3, 2007), 83 – 99.

[21] KIRNER, R., KADLEC, A., PUSCHNER, P., PRANTL, A., SCHORDAN, M., and KNOOP, J., Towards a Common WCET Annotation Language: Essential Ingredients. To appear in *Proceedings of the 8th International Workshop on Worst-Case Execution Time Analysis* (WCET 2008), (Prague, Czech Republic, July 1, 2008).

[22] LOKUCIEJEWSKI, P., FALK, H., MARWEDEL, P., WCET-driven Cache-based Procedure Positioning Optimizations, in: Proceedings of the 20th Euromicro Conference on Real-Time-Systems (ECRTS 08), Prague, Czech Republic, July 2008.

[23] LOKUCIEJEWSKI, P., FALK, H., MARWEDEL, P., and THEILING, H., WCET-Driven, Code-Size Critical Procedure Cloning, in: *Proceedings of the 11th International Workshop on Software & Compilers for Embedded Systems* (SCOPES), Munich, Germany, March 2008.

[24] NEMER, F., CASSÉ, H., SAINRAT, P., BAHSOUN, J.-P., and DE MICHIEL, M., PapaBench: a Free Real-Time Benchmark, in: *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis,* http://drops.dagstuhl.de/opus/volltexte/2006/678/.

[25] NXP Semiconductors, LPC2138 Errata Sheet, Version 1.8, July 9 2007, http://www.nxp.com/acrobat_download/erratasheets/ES_LPC2138_1.pdf.

[26] Mälardalen WCET benchmark collection, http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[27] OLIMEX Ltd., LPC-H2138 Header Board for LPC2138 ARM7TDMI-S Microcontroller, http://www.olimex.com/dev/lpc-h2138.html.

[28] OTAWA website, http://www.otawa.fr/.

[29] Philips Semiconductors, LPC2138 User Manual, Rev. 01 – 24 June 2005, http://www.nxp.com/acrobat/usermanuals/UM10120_1.pdf.

[30] PRANTL, A., SCHORDAN, M., and KNOOP, J., TuBound - A Conceptually New Tool for Worst-Case Execution Time Analysis, in: *Proceedings 8th International Workshop on Worst-Case Execution Time Analysis* (WCET 2008), Prague, 2008.

[31] PUGH, W., et al., The Omega project: frameworks and algorithms for the analysis and transformation of scientific programs, University of Maryland, http://www.cs.umd.edu/projects/omega.

[32] Rapita Systems Ltd., RapiTime On Target Timing Analysis, http://www.rapitasystems.com/.

[33] RIEDER, B., PUSCHNER, P., and WENZEL, I., Using Model Checking to derive Loop bounds of general Loops within ANSI-C applications for measurement based WCET analysis, in: *Proceedings of the Sixth Workshop on Intelligent Solutions in Embedded Systems* (WISES'08), 2008, Regensburg, Germany.

[34] ROCHANGE, C., SAINRAT, P., A Context-Parameterized Model for Static Analysis of Execution Times, in: *Transactions on High-Performance Embedded Architectures and Compilers*, 2(3), Springer, October 2007.

[35] SWEET website, http://www.mrtc.mdh.se/projects/wcet/sweet.html.

[36] TAN, L., The Worst Case Execution Time Tool Challenge 2006: The External Test, in: S*econd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation* (ISoLA 2006), 2006.

[37] Tidorum Ltd., Bound-T website, http://www.bound-t.com/.

[38] WCET Analysis: The Annotation Language Challenge website, CoSTA project, http://costa.tuwien.ac.at/languages.html.

[39] WCET Tool Challenge 2008, http://www.mrtc.mdh.se/projects/WCC08/.

[40] WCET Tool Challenge 2008 Wiki, http://www.mrtc.mdh.se/projects/WCC08/doku.php?id=start. Read-only public access. For editing access, contact the WCC'08 steering group.

[41] WENZEL, I., KIRNER, R., RIEDER, B., and PUSCHNER, P., Measurement-Based Timing Analysis, in: *Proc. 3rd Int'l Symposium on Leveraging Applications of Formal Methods, Verification and Validation* (ISoLA'08), to appear, 2008, Porto Sani, Greece.

[42] WILHELM, R., ENGBLOM, J., ERMEDAHL, A., HOLSTI, N., THESING, S., WHALLEY, D., BERNAT, G., FERDINAND, C., HECKMANN, R., MITRA, T., MUELLER, F., PUAUT, I., PUSCHNER, P., STASCHULAT, J., and STENSTRÖM, P., The worst-case execution time problem — overview of methods and survey of tools, in: ACM Transactions on Embedded Computing Systems, Volume 7, Issue 3, April 2008.