A Survey of WCET Analysis of Real-Time Operating Systems*

Mingsong Lv[1], Nan Guan[1], Yi Zhang[1], Qingxu Deng[1], Ge Yu[1], Jianming Zhang[2]

[1] Northeastern University, Shenyang, China P.R.
[2] China Center for Information Industry Development

Abstract

Timing correctness of hard real-time systems is guaranteed by schedulability analysis and worst-case execution time (WCET) analysis of programs. Traditional WCET analysis mainly deals with application programs and has achieved success in industry. Timing analysis of application programs along cannot guarantee correctness of complete systems consisting RTOS. WCET tools designed for application program analysis have been applied to analyze RTOS routines by several research groups, but poor WCET estimations have been reported. Timing analysis of real-time systems considering both applications and RTOS has not been fully studied. So we intend to give a survey of related work on WCET analysis of RTOS. By summarizing previous work, challenges of WCET analysis of complete real-time systems are presented, and some possible further research potentials are unleashed.

1 Introduction

Hard real-time systems are those systems the tasks of which must meet their deadlines, otherwise, there will be disastrous consequences. Timing correctness of hard real-time systems is traditionally guaranteed by separate schedulability analysis and worst-case execution time (WCET) analysis. WCET analysis is used to obtain execution times of tasks, and schedulability analysis use these results to decide whether all the tasks in the system are schedulable. Traditional WCET analysis mainly focuses on application programs and has achieved success in industry. While real-time systems are composed of both applications and RTOS, and the timing properties of the system are decided by both parts. So in order to obtain usable WCET estimations for real systems, timing analysis should be performed on both application programs and RTOS. Interdependence and interactions of these two parts should also be considered.

Several groups have conducted research on WCET analysis of RTOS using static analysis tools designed to analyze application programs. Unfortunately, almost all the projects reported difficulties in analyzing RTOS. Lots of problems that may lead to large overestimation are unveiled. For example, loops are much harder to bound in RTOS analysis since most loop bounds are decided by runtime properties of the tasks. The reason behind is that very little information on application programs is considered in the analysis of RTOS. Moreover, traditional WCET analysis assumes non-interruptible programs, but the existence of RTOS introduces task preemptions, which makes this assumption invalidated in real systems. Related research shows that on complex CPUs (e.g. those exhibit timing anomalies) separate schedulability and WCET analysis may even be unsafe since task switching may prolong the execution time of the tasks. This is rather undesirable in safe-critical hard real-time systems.

Results reported so far show that existing WCET tools or techniques designed to analyze application programs cannot properly handle RTOS, so it is highly desirable to develop new techniques and tools for the timing analysis of RTOS. In this paper, we first give a survey of related research practices on timing analysis of RTOS using static methods. Then the problems encountered are summarized and possible new challenges are given. Although some of the challenges have been identified in WCET analysis of applications, they are generally circumvented before. The need for timing analysis of RTOS brings the problems again onto the stage. We believe there should be more efforts on WCET techniques and tools to meet the challenges presented by timing analysis of RTOS, which will improve the usability of WCET analysis [14] in real systems.

The rest of the paper is organized as follows. Section 2 gives some background information on WCET analysis. Related research projects are detailed in Setction 3. In Section 4, we summarize the problems reported in the research

*This work was partially sponsored by the National High Technology Research and Development Program of China (863 Program) under Grant No. 2007AA01Z181 and the Cultivation Fund of the Key Scientific and Technical Innovation Project of Ministry of Education of China under Grant No. 706016.

practices and show what should be enforced in WCET analysis. The paper is concluded in Section 5.

2 Overview of Timing Analysis Techniques

The results of WCET analysis are required to be *safe* and *accurate*: no actual execution of the program should exceed the estimated time, and the estimation should be as close as possible to the real maximal execution time of the program. Soft real-time systems do not always have safety requirements, but hard real-time systems require that the WCET of programs should never be underestimated. The accuracy of the estimation can affect the quality of schedulability analysis: too pessimistic estimations lead to over-design and very low task accept ratio.

There are two major types of timing analysis techniques: *static analysis* and *dynamic analysis*. Static analysis derives the WCET of a program by statically analyzing the behavior of the code without actually executing it. Dynamic analysis determine program timing by measurements: the program is executed many times with different inputs and the execution time is measured by either software methods or hardware instruments, such as oscillators and logic analyzers.

Theoretically, static analysis can either explicitly or implicitly consider all possible executions of the program, so safety is guaranteed. But the problem is that in presence of complex hardware the program state space is too large to handle. Since the program is not actually executed, a safe abstract hardware model is always needed, but such models are usually hard to construct. Obviously, dynamic analysis requires no such abstract hardware models. The problem of dynamic analysis is that safety of the results is not guaranteed, because it is generally impossible to cover all program states by execution.

Traditionally, timing analysis of RTOS is performed by dynamic analysis techniques. But in hard real-time systems, safety of the WCET results is mandatory, so measurement-based methods are not suitable for such systems. That is why multiple research groups tried to analyze RTOS via static methods.

Static analysis usually works as follows. The analysis starts from control flow analysis that reconstructs the Control Flow Graph (CFG) from the binary code or the source code of the program. Then a processor behavior analysis follows to estimate the execution time of each basic block in the CFG running on some specific hardware. The abstract hardware model is used in this step. Finally, the estimation is calculated by finding the path that leads to the longest execution time, given the results obtained from the first two steps. Possible techniques to search this path are Integer Linear Programming (ILP) [3, 18, 19], model checking [22, 23, 36], and tree-based calculation [1, 10].

The above content is to provide necessary background information on WCET analysis for better understanding. Interested readers can refer [35] for an excellent survey of general WCET analysis techniques and tools.

3 Related Research on WCET Analysis of RTOS

This section surveys related research on WCET Analysis of RTOS. Research from six groups are introduced and the problems reported and the solutions are presented.

3.1 Static Timing Analysis of RTEMS

Colin and Puaut are the first group to conduct research on static WCET analysis of RTOS [11]. The RTEMS [2] real-time kernel was analyzed using HEPTANE [10] which was developed by the same group. HEPTANE is a tree-based WCET analyzer with limited capabilities on micro-architecture modeling. At the time of the research, RTEMS had 85 system calls provided by 17 modules, but only 12 system calls were analyzed. The analyzed system calls consisted of 91 source files and contained 14,532 lines of code. Lots of problems were reported on analyzing RTOS, which are given as follows.

The first problem Colin encountered in practice was unstructured control flow, such as multiple loop exits due to the use of `goto` statement within the loop body. Almost all tree-based WCET analyzer suffer this problem, while IPET-based tools can easily handle it. This problem is specific to the analyzer instead of the analyzed program. Colin tackled this problem by re-writing the loops.

The second problem was from dynamic function calls that are implemented through function pointers. Dynamic function calls are a notorious problem for static WCET analysis, because the called function is known only at run-time. In RTEMS, dynamic function calls were introduced by user extensions, API extensions and the calling of architecture-dependent hooks. If such function calls can be fixed at design time, then the problem disappears; otherwise, additional methods are required to deal with dynamic function calls. In Colin's work, dynamic function calls were replaced by static ones.

The third type of problems came from the determination of loop bounds. WCET tools usually require the user to give loop bounds manually. This works well in the analysis of application programs, but awkward for RTOS analysis, because most loop bounds are related to the RTOS's dynamic runtime behaviors. For example, the scheduler often iterates a run-queue to find the task with the highest priority to be scheduled, so the loop bound is determined by the maximum number of running tasks in the system. If the user is ignorant of this information, he has to assume the extreme case where there are MAX_TASK tasks in the system (MAX_TASK specifies the maximum number of allowable tasks), which leads to big pessimism in the estimated results. The problem existed in management of names in string and user extensions, heap management and task management of RTEMS. Loops were bounded manually in Colin's work by a close investigation of the codes.

The most difficult problem was that the body of the scheduler is a loop, the loop count of which is affected by interrupts issued during the run time of the scheduler. For example, in the first iteration of the loop, there may be a new interrupt appended, which leads to a future iteration to handle the interrupt. In the worst case, a new interrupt comes and is appended in each iteration, so it is not trivial to bound the loop. Colin pointed out that this problem is feasible if enough information on the interrupts are available.

Blocking system calls are hard to analyze and this is the killer to most WCET tools. Colin also pointed out that a static WCET analyzer just takes the `Context_Switch` function as a normal function and ignores the hardware behaviors specific to context switches. Interrupt routines themselves are easy to analyze, but they may affect the loop iterations of the scheduler. Colin required to ensure no blocking system calls, and the context switch time related to cache and pipeline behaviors was not well handled.

Putting them all together, an average of 86% overestimation was reported in Colin's work. This result is much worse than those obtained in the analysis of application programs. Although almost no really new analysis technique was given, Colin's work paved the way for WCET analysis of RTOS using static methods and gave lots of insights on this research topic.

3.2 Combined Schedulability and WCET Analysis at Saarland University

In [33], Schneider proposed a comprehensive framework considering both schedulability analysis, WCET analysis and their interdependence, which was built on an argue that the precision of WCET analysis of RTOS can be improved by considering the applications, and vise versa. Some of the difficulties encountered in RTOS analysis were identified to support this argue. The first type of problems came from system calls: if the calling parameters, calling context, or even the calling history is considered, we can obtain a more precise estimation of the WCET of the system calls, since these information can help to identify system modes and infeasible pathes. Another problem was cache state changes due to inter-task cache replacement, and this side effect was highly related to task preemption. Static system parameters determined by applications are also needed. The precision of analyzing the WCET of applications can also be improved by considering positive effects from RTOS. So Schneider proposed that it is desirable to analyze both RTOS and applications in an integrated framework considering their interdependence. A high-level view of the system to consider both schedulability and WCET analysis is also important to the quality of RTOS analysis.

In [31] and Schnieder's Ph.D. dissertation [32], combined schedulability and WCET analysis was discussed. Schneider first demonstrated the existence of timing anomalies and domino effects in the PowerPC 755 processor, and these effects could be triggered by task preemptions. Then he proofed that in presence of timing anomalies traditional separated schedulability and WCET analysis might be unsafe, and if a safe margin was added to context switch overhead, the results might be very pessimistic. Timing anomalies also make Deadline Monotonic non-optimal for independent periodic task sets. Against this background, a combined schedulability and WCET analysis framework was proposed, which could give safe timing predictions and schedulability results on complex microprocessors exhibiting timing anomalies and domino effects.

The higher level of the analysis framework was Response Time Based (RTB) schedulability analysis which made almost no difference from traditional RTB analysis methods: all execution time information on tasks and RTOS routines were needed to calculate the response times. The major difference existed in the estimation of task WCETs. For each task, all the interfering jobs were identified and the interfering information was provided to the WCET analysis where the WCET of the task was calculated considering task-switching-triggered timing anomalies and domino effects. The cache effects and pipeline effects were calculated as part of the execution time of the task instead of context switch cost. The framework was not implemented in Schneider's dissertation, so no concrete evaluations on the degree of automation were reported.

3.3 Static Timing Analysis of the OSE Kernel

The WCET analysis of the OSE kernel was conducted by Martin Carlsson [8, 9] and Daniel Sandell [28, 29] conjunctively. Carlsson's work centered on timing analysis of the Disable Interrupt (DI) regions of the OSE kernel [12] running on an ARM9 processor. The purpose of his research was twofold: timing analysis of DI regions can help the OS vendors to optimize their OS for better responsiveness; this work can also provide valuable information about real system codes for WCET researchers.

The analysis tool adopted was SWEET [1], extended with a "frontend" to extract DI regions from the object code. The OSE kernel contained more than 1,200 DI regions, and timing analysis was performed on 612 DI regions that could be identified by the analysis tool. It was found out that 554 regions contained no more than three basic blocks, and loops were found in about 5% regions, which means the control flow of the DI regions are quite simple. Poor estimations were reported due to the lack of cache modeling in the analysis tool and the difficulty in bounding the loops.

Carlsson made several suggestions on static timing analysis of RTOS according to his experience. First, there is a need to make relevant high-level information available on the object code level, because high-level information on loop bounds, OS modes and task information greatly affect the precision of the estimated results. Work on data flow analysis and compilers should be enforced. Traditional analysis techniques on bounding loops seems insufficient, so a more general language to express constraints on value ranges would be useful. Second, absolute WCET estimations are of less interest for timing analysis of RTOS. There is a great potential in improving the degree of automation in the analysis of RTOS.

Sandell used the aiT tool [3] to analyze the OSE kernel running on an ARM7TDMI processor. Two main objectives of his research were to find out how hard it was to analyze typical operating systems codes, and how compiler optimizations affected the manual labor needed to perform an accurate WCET analysis, where the former problem is our focus. Four system calls on memory allocation and interprocess communication were analyzed.

Sandell encountered three major problems in the analysis. The first one was the execution times of the system calls highly depend on system parameters, such as the number of signal buffers or maximum message sizes. The second problem was some loop bounds were determined by system parameters at runtime, so even the powerful aiT tool was used, many loops could not be properly bounded. The third problem was the execution times varied a lot according to different system modes (normal mode or error handling), but most of the time the system worked in some typical scenarios. Lots of user intervention was required throughout the analysis. It was concluded that the static WCET analysis techniques were not mature enough to fully automate the timing analysis of RTOS on a ”one-click-analysis” basis. Sandell also pointed out that absolute WCET bounds were not appropriate for RTOS codes, and the constant time assumed for context switch might be too pessimistic. A series of case studies of applying WCET analysis in industrial settings confirmed the conclusions of Sandell's work [15].

3.4 Timing Analysis of L4 Kernel at NICTA

A team at NICTA Australia, led by Stefan Petters, performed WCET analysis of the L4 real-time kernel [27] (developed by NICTA and still evolving) with the objective of exploring the degree of automation in WCET analysis of RTOS codes. The analysis tool uses a hybrid design with a tree representation of the control flow of the program and the execution time of each basic block obtained by measurement [30]. Efforts were made to analyze the whole L4 kernel and some analysis obstacles were reported in their paper [34]. Around ten challenges were reported, but we are trying to summarize them according to the fundamental source of difficulties.

The first challenge came from the code structure. Non well-structured codes, such as irreducible loops (mainly due to the use of `break` or `goto` statements within the loop), were a big problem since the WCET tool they were using is tree-based. This problem can be resolved by virtually unrolling the loop or duplicating code to obtain a reducible structure. Assembly codes in the L4 kernel also introduced similar problems. Code structure problems were resolved manually in Petters' work.

Another problem is the indexed jumping due to compiler optimization: the compiler creates a hash table of all the `case` addresses and make an indirect jump to these options. Efforts are needed to identify the jump tables. The return address in the ARM architecture is stored in a specific register, so in order to reconstruct the call trace, a tracer that can analyze register contents was designed.

A third major problem is dynamic function calls. For example, the interrupt vector table is such a construct. Manual analysis is performed in Petters' work. A special case is that function pointers are indexed in a loop. For such situations, loop unrolling is useful to flatten the control flow and attach to each loop iteration a dedicated function call.

Petters also considered context switches in measurement of the execution time of the basic blocks, but the interaction with the scheduler is not discussed in his paper. Petters also encountered problems in memory allocation, which is quite similar to that reported in Colin's work. They suggested proper design at the RTOS side to resolve these problems.

Expertise in the RTOS to be analyzed and the WCET tool adopted is crucial throughout the analysis. the L4 kernel will be augmented to a multiprocessing kernel in the future, this will pose new challenges to WCET analysis.

3.5 Research on Predictable Architectures at TUWien

Traditionally, timing analysis of hard real-time systems assumes a hierarchical model that separates the low-level task timing issues from the high-level real-time scheduling problem. This worked well for systems running on simple processors in the past, but yields bad results with the adoption of more and more complex hardware in hard real-time systems. In [25], Puschner investigated how to design a system in a way that composable timing analysis can still be used without laborious work on designing fancy WCET tools or introducing too much pessimism into the analysis.

The author mainly discussed timing effects due to complex hardware architectures and task scheduling. Although this work did not directly target at WCET analysis of RTOS, it unveiled some problems that needs considering.

The author defined the biggest obstacle to composable WCET analysis as "side effects": task interactions that cannot be traced back to the interfaces between tasks and their environment. For example, task A is preempted by task B according to some scheduling policy, and B's execution replaces A's data in the cache, which affects the execution time of the remainder of A. This effect cannot be traced back to any task interface.

In simple hardware architectures, the execution times of tasks may vary because there are instructions with the execution cycles of which determined by the operands. Different task data inputs may also lead to execution time variations. In complex hardware, side effects are harder to predict. Due to complex hardware features, different instances of the same task may have different execution times. Does this variation will eventually stabilize? After how many issues will the execution time converge? These questions are very hard to answer. When scheduling exists, even if preemption is turned off, the execution time of a task may also vary since different tasks may execute alternatively, which creates different start states for each task instance. If preemption exists, the problem will be more severe, since hardware states will change at the preemption points. Other problems also come from out-of-order pipelines that can introduce timing anomaly [21, 26], and the problem state space will be prohibitively large.

Since multi-core processors are gaining popularity in embedded systems, new architectural features will pose new challenges to WCET analysis. One problem is shared cache: if two tasks on two different cores share the same cache, it is hard to bound the effects of mutual replacement of cache contents. Other shared resources, such as shared bus, have similar problems. Another problem is SMT, which enables multiple tasks on the same core to share the function units or pipelines at instruction level. Analyzing such inter-task interferences is not trivial.

Puschner took one extreme to tackle the above problems. The basic idea is try every possibility to avoid side effects that make task execution unpredictable. Four techniques were presented: (1) Using single-path programming in all tasks [24] to reduce intra-task side effects; (2) Forcing execution of a single task/thread per core to reduce inter-task interference; (3) Using simple hardware with in-order pipelines to reduce the state space; (4) Statically scheduled accesses to shared memory to reduce side effects due to resource sharing. Khyo and Puschner later designed a platform [17, 16] with the hardware and RTOS implementing the above design philosophy.

Puschner's platform is predictable at clock cycle level, but this is not achieved without sacrifice. In single-path codes, both paths of an "if-then-else" construct must be executed, the system has to do lots of unnecessary computation. The system requires that all the context switches and IPCs are scheduled offline, this is too restricted for those real-time systems that have high responsiveness to the environment which cannot be pre-determined. Puschner's work demonstrates one extreme: ultimate predictability can be achieved at the design stages on both hardware and software, and this greatly simplifies analysis efforts since most of the unpredictable timing features are eliminated.

3.6 Miscellaneous

Aissa et al in [4, 5] proposed a distributed WCET computation scheme for smart card systems. Smart card systems are interesting: the source codes are compiled into some intermediate form which is then downloaded to the smart card and compiled on-line into native codes before execution. In this case, the WCET of the programs can only be determined on-card, which poses great challenge to the analysis complexity. The main idea is to offload the burden of on-card analysis as much as possible. The problem is tackled in a distributed manner: an off-card parser flattens the CFG into a weighted tree, and then the tree is sent to the card and analyzed by the compiler. Loops are bounded by user annotation, and the bounds are transformed into codes that ensure the execution of each loop will not exceed its bound. The difficulty mainly comes from the peculiarities of the smart card system, Aissa's work was not directly related to WCET analysis of RTOS discussed throughout this paper.

4 Summarization of Problems and New Challenges

In this section, we summarize and classify the problems on static timing analysis of RTOS, and present our vision on the challenges that should be addressed in future research.

4.1 Summarization of Problems on Static Timing Analysis of RTOS

The problems encountered in the above research practices on static timing analysis of RTOS are summarized in Figure 1, and they are roughly classified into three categories: (1) Problems due to program features; (2) Problems due to the lack of application information; (3) Problems due to task switching and inter-task interference.

Problems such as irreducible program structures are not critical to WCET analysis of RTOS, since most of such problems do not exist if you choose a proper WCET tool that is flexible in handling complex control flows.

	Colin	Schneider	Sandell	Petters	Puschner
RTOS Analyzed	RTEMS	OSE	OSE	L4	
Analysis Tool	HEPTANE	aiT	aiT/SWEET	Petters' Tool	
Average Overestimation	86%	n/a	n/a	n/a	
Problems Due to Program Features					
Irreducible Program Structure	P2			P2	
Indexed Jumping				S	
Problems Due to Lack of Application Information					
Hard to Bound Loops Due to Runtime Properties	P2,3		P2,3	N	
Dynamic Function Calls	P4			P4	
Blocking System Calls	N				
Lack of Knowledge on System Call Contexts		P3			
Lack of Knowledge on RTOS Running Mode			P2		
Problems Due to Task Switching and Inter-Task Interference					
Timing Effects Due to Task Switching		P1			P4
Timing Anomalies Due to Preemption		P1			P4
Inaccurate Execution Time of Context Switches	N	S	N		P4
Inter-Task Interference Due to Resource Sharing on Multicores				N	N

* "S": the problem is properly solved

* "N": the problem is circumvented in related research

* "P": the problem is partially solved, but needs further development. Possible problems may be low scalability of the analysis (P1), too much user intervention required (P2), low quality of the results (P3), or the adopted techniques are too restrictive (P4)

Figure 1. A List of Analysis Problems Reported in Research Practices

Problems due to the lack of application information greatly affect analyzability and the precision of the results. The biggest problem reported by many groups is the difficulty in bounding loops, since the loop bounds in RTOS has very close relation to the runtime properties and system parameters. Dynamic function calls and blocking system calls should be classified as program features which are generally hard to tackle in traditional WCET analysis. But if we can extract more application information on function/system calls and communicate it to RTOS analysis, the behavior of such program features may be bounded. And if more information on call context and RTOS working mode is given, the precision of the analysis can be further improved. Problems of existing techniques are that lots of user intervention is required to give these information, and poor results are obtained due to this type of problems.

Traditionally WCET analysis assumes uninterrupted program execution. This is not the case in multi-tasking systems running an RTOS. In presence of task switching, the execution time of a task also depends on the behaviors of the interfering tasks. The timing effects due to task switching must be safely bounded in the analysis. Schneider tried to conduct detailed analysis on complex hardware in each possible preemption point. Experiments were only done on simple programs, but there will probably be scalability problems when analyzing complex programs. Puschner took the other extreme by designing a predictable RTOS to circumvent all these problems. We believe his approach is effective but too restrictive. In the multi-core era, tasks on different cores may share lots of hardware resources. Resource sharing deteriorates inter-task interferences and the execution time is much harder to estimate. This problem was touched by Petters and Puschner respectively, but neither of them gave any solution.

4.2 Challenges on Static Timing Analysis of RTOS

Although different research groups put great efforts on static timing analysis of RTOS, we can see from the problem summarization that most of the identified problems have not been solved properly. New challenges emerge in many aspects of the analysis.

4.2.1 Does Single WCET Value Suffice?

WCET analysis of application programs is mature, and for many years the results that WCET analysis offers is single absolute value. In related research practices, we hear a strong voice for parametric WCET analysis, in which the results are given as a formula instead of a single value. Different from application programs, the execution time of RTOS highly depends on static or runtime system parameters, so a single value can never give a tight bound for all possible system configurations.

Lisper in [20] proposed the idea of parametric WCET analysis, and his work recently evolved to handle complex architectures [6]. In [7], Bygde and Lisper demonstrated that this technique is currently applicable to small programs. Since ILP-based method dominates the WCET tools from both academia and industry, we believe that Lisper's parametric ILP could be one of the promising techniques to obtain parametric formulas. Lisper's work mainly centered on parameterizing loops, but in RTOS analysis, lots of system parameters, besides loop bounds, affect the execution time of the system calls, so they should also be characterized as parameters. Research on parameterizing the system calls across a broader spectrum should be developed. Since currently parametric WCET analysis cannot handle large programs, efforts on optimization techniques to efficiently obtain the formulas for this specific type of problems (WCET analysis) are key to the usability of parametric techniques in WCET analysis of RTOS.

4.2.2 WCET Analysis Considering Both Applications and RTOS

Related research practices show that the analyzability and the precision of the RTOS analysis results can benefit from information provided by the applications. Examples of such application information include the number of tasks, communication behaviors, memory allocation policies, etc. One critical problem is "How to make application information available to the RTOS analyzer?" Up till now, users play the role to communicate application information to WCET analysis of RTOS, but this is obviously not desirable. To automate this procedure, new techniques should be developed. First, the techniques must parse the source code of applications and RTOS to extract useful information. Then they must be able to correctly map the information from the source code level to the binary code level. Mapping of flow facts is one of the key issues in WCET analysis, but it still needs further development.

4.2.3 Combined Schedulability and WCET Analysis

Traditional real-time analysis adopts the separate framework - performing WCET and schedulability analysis in two separate steps, with the assumption that the WCETs of tasks are fixed and never affected by scheduling behaviors. Obviously in a multi-tasking real-time system, it is not practical to neglect the effect of inter-task interferences. The situation is even more severe in multi-core systems since tasks on different cores interfere with each other in a more fine-grained manner. Schneider demonstrated that on complex hardware, separate schedulability and WCET analysis may be unsafe due to timing anomalies and domino effects. If one still want a separate analysis framework, large pessimism has to be introduced into the WCET of tasks or RTOS. In [13], a combined analysis framework with mutual communication between schedulability analysis and WCET analysis is proposed, but neither detailed information on how it works nor the quality of results is given. We believe that schedulability and WCET analysis should be performed in an integrated and interactive manner when analyzing complete real-time systems running on complex hardware. The biggest problem is the state space explosion due to combined analysis. So efficient abstraction techniques to reduce the state space in combined analysis should be developed and evaluated for practical use in analyzing large systems.

4.2.4 Raising the Degree of Automation in WCET Analysis of RTOS

Almost all the related research practices reported low degree of automation in analyzing RTOS. Users worked laboriously revising irreducible source codes, bounding the loops, resolving dynamic calls, etc. This has become the most notorious problem in WCET analysis of RTOS. It is true that this issue is not orthogonal with the other problems, but WCET tool designers must always keep the issue of "automation" in mind when trying to tackle any of the problems, for the degree of automation is the largest factor that affects the usability of a WCET tool.

4.2.5 Managing Analysis Complexity in the Multi-core Era

Multi-core has become the inevitable architecture of future processors. Caches are often designed as shared resource due to design flexibility, data sharing and coherency considerations. Fine-grained accesses to shared resources lead to complex inter-task interference and unmanageable analysis complexity - this is the biggest challenge to timing analysis posed by multi-cores. To tackle these problems, performance isolation techniques (e.g. cache partitioning) and predictability enforcing techniques (e.g. cache locking) have been applied to enable resource sharing in a controlled manner, but overall system performance is sacrificed. We believe that pure analysis techniques with safety as the unique goal is inadequate, both design and analysis techniques should be adopted to ensure that on one hand the analysis complexity is manageable, and on the other hand the horsepower provided by multi-cores is least sacrificed. Research on timing analysis of multi-core real-time systems is becoming hot, but far from mature.

5 Conclusion

In this paper, we conduct a survey of the research practices in static timing analysis of RTOS. Lots of problems were reported by the practices, which shows that WCET tools for application program analysis cannot be directly used to RTOS analysis without further development. Critical problems include usability of single-value WCET results, lack of application information on RTOS analysis, and neglecting inter-task interferences. These problems pose great challenge to the application of WCET analysis in real systems. The emergence of multi-core architecture also brings new challenges to WCET analysis of both applications and RTOS. Although some of the challenges may be not really new, they are generally ignored due to analysis complexity in previous research. We hope the survey can help to clarify the challenges in timing analysis of RTOS, and to motivate new analysis techniques to crack these tough problems to improve the usability of WCET analysis in real systems.

References

[1] *http://www.mrtc.mdh.se/projects/wcet/sweet.html*.

[2] http://www.rtems.com.

[3] AbsInt. The absint page. *http://www.absint.com*.

[4] N. B. H. Aissa, D. Deville, and G. Grimaud. A distributed wcet computation scheme for smart card operating systems. *In WCET 2004*.

[5] N. B. H. Aissa, G. Grimaud, and V. Benony. Bringing worst case execution time awareness to an open smart card os. *In RTCSA 2007*.

[6] S. Altmeyer, C. Hümbert, B. Lisper, and R. Wilhelm. Parametric timing analysis for complex architectures. In *RTCSA 2008*.

[7] S. Bygde and B. Lisper. Towards an automatic parametric wcet analysis. In *WCET 2008*.

[8] M. Carlsson. Worst case execution time analysis, case study on interrupt latency for the ose real-time operating system. *Master Thesis of Royal Institute of Technology*, 2002.

[9] M. Carlsson, J. Engblom, A. Ermedahl, J. Lindblad, and B. Lisper. Worst-case execution time analysis of disable interrupt regions in a commercial real-time operating system. *In 2nd International Workshop on Real-Time Tools*, 2002.

[10] A. Colin and I. Puaut. A modular and retargetable framework for tree-based wcet analysis. *13th Euromicro Conference on Real-Time Systems*, 2001.

[11] A. Colin and I. Puaut. Worst-case execution time analysis of the rtems real-time operating system. *13th Euromicro Conference on Real-Time Systems*, 2001.

[12] Enea. Enea embedded technology page. *www.enea.com*.

[13] C. Ferdinand and R. Heckmann. Worst-case execution time - a tool provider's perspective. In *ISORC 2008*.

[14] J. Gustafsson. Usability aspects of wcet analysis. In *In ISORC 2008*.

[15] J. Gustafsson and A. Ermedahl. Experiences from applying wcet analysis in industrial settings. *In ISORC 2007*.

[16] G. Khyo, P. Puschner, and M. Delvai. An operating system for a time-predictable computing node. *The 6th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, pages 150–161, 2008.

[17] R. Kirner and P. Puschner. Time-predictable task preemption for real-time systems with direct-mapped instruction cache. *In ISROC 2007*.

[18] X. Li, Y. Liang, T. Mitra, and A. Roychoudury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1-3):56–67.

[19] Y.-T. S. Li, S. Malik, and A. Wolfe. Cinderella: A retargetable environment for performance analysis of real-time software. In *Euro-Par 1997*.

[20] B. Lisper. Fully automatic, parametric worst-case execution time analysis. In *WCET 2003*.

[21] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *The 20th IEEE Real-Time Systems Symposium*, 1999.

[22] M. Lv, Z. Gu, N. Guan, Q. Deng, and G. Yu. Performance comparison of techniques on static path analysis of wcet. In *The 5th International Conference on Embedded and Ubiquitous Computing*, 2008.

[23] A. Metzner. Why model checking can improve wcet analysis. In *CAV*, pages 334–347, 2004.

[24] P. Puschner. Transforming execution-time boundable code into temporally predictable code. In *SDPES*, 2002.

[25] P. Puschner and M. Schoeberl. On composable system timing, task timing, and wcet analysis. In *WCET 2008*.

[26] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. In *WCET*, 2006.

[27] S. Ruocco. Real-time programming and l4 microkernels. In *In Proceedings of the 2006 Workshop on Operating System Platforms for Embedded Real-Time Applications*, 2006.

[28] D. Sandell. Evaluating static worst-case execution-time analysis for a commercial real-time operating system. *Master Thesis of Malardalen University*, 2004.

[29] D. Sandell, A. Ermedahl, J. Gustafsson, and B. Lisper. Static timing analysis of real-time operating system code. *In 1st International Symposium on Leveraging Applications of Formal Methods*, 2004.

[30] S. Schaefer, B. Scholz, S. M. Petters, and G. Heiser. Static analysis support for measurement-based wcet analysis. In *WiP Session of RTCSA 2006*.

[31] J. Schneider. Cache and pipeline sensitive fixed priority scheduling for preemptive real-time systems. *21st IEEE Real-Time Systems Symposium*, 2000.

[32] J. Schneider. Combined schedulability and wcet analysis for real-time operating systems. *Ph.D. thesis of Saarland University, Germany*, 2002.

[33] J. Schneider. Why you can't analyze rtoss without considering applications and vice versa. *2nd International Workshop on Worst-Case Execution Time Analysis*, 2002.

[34] M. Singal and S. M. Petters. Issues in analysing l4 for its wcet. *Proceedings of the 1st International Workshop on Microkernels for Embedded Systems*, 2007.

[35] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *Trans. on Embedded Computing Sys.*, 7(3):1–53, 2008.

[36] S. Wilhelm. Efficient analysis of pipeline models for wcet computation. In *WCET 2007*.