ISBN 978-981-11-0008-6
Proceedings of 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016)
Tokyo, Japan, 17-19 June, 2016, pp. 13 - 17
doi: 10.18178/wcse.2016.06.003

Timing Analysis of Parallel Embedded Real-Time Software–Practical Experiences

Muhammad Waqar Aziz and Syed Abdul Baqi Shah+

Science and Technology Unit, Umm Al-Qura University, Makkah, 21955, Saudi Arabia

Abstract. The knowledge about the Worst-Case Execution Time (WCET) of a program is fundamental to the successful design of embedded real-time software. The WCET analysis of embedded software is also needed for tasks scheduling, understanding the system and to guarantee its behavior. The conventional WCET analysis deals with the sequential code running on uni-processors. However, the increasing use of multi-core processors in embedded real-time systems; the WCET analyses face new problems. This article reports our experiences of performing the WCET analysis of Parallel Embedded Software (PES) running on multi-core hardware. The main objective was to investigate the way WCET estimates of PES can be computed both statically and dynamically. The experiences reported in this article include the challenges faced, possible solutions to these challenges and the workarounds developed. This article also provides observations on the benefits and drawbacks of deriving the WCET estimates and sets path for future research in this direction.

Keywords: embedded real-time software, worst-case execution-time analysis, parallel computing.

1. Introduction

Embedded software mostly has real-time nature, i.e., they need to interact with their environment within stringent deadlines. It means that they must operate in a timely manner to ensure their correct working, since in some cases logically correct results are of no use if delivered late. This makes the temporal verification of embedded software as important as their functional one. To guarantee the correct timing behavior, a schedulability analysis is performed that verifies all tasks can meet their deadlines at runtime. This requires knowledge about the Worst Case Execution Time (WCET) of the individual tasks. There are two main types of WCET analysis [1]: static and dynamic. In static analysis the program flow is analyzed without executing it, whereas in dynamic analysis the execution time of the program is measured by executing it on actual hardware or a simulator.

Unfortunately, the steps of static WCET analysis [1] work adequately for sequential programs running on single-core hardware architectures, but are challenged in the parallel-computing world. First, programs do not execute as a stream of sequential instructions, so conventional control and data-flow analyses must be updated to take into account the inherent concurrency in parallel programs. Second, hardware no longer has bounded timing behavior because of inter-thread interferences making hardware modeling impossible or extremely hard in the best case. Third, calculation techniques for WCET analysis are suited for additive sequential models where the overall WCET is derived from smaller execution times of constituent segments by adding them together reflecting their non-parallel execution.

This research work is aimed at investigating the cutting-edge solutions for timing analysis of Parallel Embedded Software (PES) running on multi-core architectures. This article reports our practical experiences of performing WCET analysis of PES. This includes (1) a high-level (code-level) static flow analysis of the PES that identifies the execution behavior of the PES, and (2) a low-level dynamic analysis of PES that

+ Corresponding author. Tel.: +966-58-392-7245.
E-mail address: sashah@uqu.edu.sa

measures its execution time to observe its timing behavior. This article also reports the pitfalls, the workarounds we made, our findings and observations. In addition to providing useful suggestions and recommendations, this article also provides novel ways to capture the timing properties of parallel executing threads such as threads execution times, inter-thread flow information, and so on.

2. Approach and Scope

The two major themes of this work are flow-analysis and dynamic analysis of PES. Therefore the philosophy, used in this work, was to isolate research on high-level flow-analysis from research on low-level timing analysis initially, then combine them later in the calculation phase. For this reason, during the flow analysis of parallel programs, any execution-time variations were abstracted away. This was achieved by assuming that the execution time of a section of code is k if it contains k object-code instructions. The high-level flow-analysis was performed by statically analyzing the concurrent behavior of parallel programs at the code-level. The flow-analysis, in this work, was related to finding the thread's information and the execution count of a program segment. In contrast, the dynamic analysis involved the measurement of execution times of PES using the program execution traces generated through Gem5 architecture simulator [2]. The start and end of thread execution times were extracted that allowed to derive the execution times of the individual threads as well as the entire application.

This work is focused on PES developed using POSIX threads, which is the widely-used industry standard for developing parallel programs. The POSIX standard provides the explicit characterization of thread control (creation and join) and synchronization through mutexes and barriers. The parallel software, in this work, refers to a single program where a task is decomposed into threads that run in parallel to achieve high throughput and low response time. This work does not deal with system-level timing analysis, which involves combined execution of different programs. Thus, the issues related to massive parallelism, such as communication costs due to networking of different nodes, are beyond the scope of this work. Similarly, by low-level analysis we mean finding the actual execution times of the PES in number of clock cycles, which should not be confused with the micro-architecture analyses, such as analysis of contention effects caused by parallel hardware. As this work is related to exploring the ways of WCET analyses, calculating the WCET estimates through static or hybrid analysis method is not possible at this stage.

3. Timing Analysis of Parallel Embedded Software

This section details how the timing analysis was performed and presents the outcomes. The ParMiBench benchmark suite [3] – an open-source set of embedded parallel programs, is used as example PES.

High-level Static Flow Analysis. Flow analysis is the first phase of WCET analysis responsible for deriving high-level timing information which is required by the later phases, such as processor-behavior analysis and the calculation phase. The flow information about ParMibench and the way it was collected is detailed below.

Thread Basic Information: includes finding the number (and times) of threads created and the function passed to them. This information is useful in thread scheduling and their mapping to cores. In ParMiBench, mostly the user is provided the option to enter the number of threads. Having a fixed number threads makes the PES more predictable. In some benchmarks, (e.g., Stringsearch) threads are created only once, while in others (e.g., Susan), threads are created twice or more. As a program spend more of its execution time in loops, all the loops present in the function passed to the thread were analyzed and the one with more iterations was selected. The identified code segment was then instrumented with counters to find the number of times the code segment would execute. This execution count information is later required in the calculation phase of the static analysis to compute the WCET estimates.

Task Information: analyzes task granularity and work distribution in terms of its decomposition among threads. In ParMiBench, tasks are mostly decomposed in a coarse-grained fashion, resulting in low thread synchronization and communication. The fine-grained task decomposition would result in increased synchronization and communication overhead. In such case, the time spend in synchronization and communication should be analyzed and added to the overall WCET. Further ParMiBench uses static load balancing, i.e., the work is equally distributed among threads to avoid load unbalance. The synchronization related information was collected using the sync constructs present in the parallel code, e.g., barriers, locks, condition variables, and joins.

Dynamic Analysis. The dynamic analysis is concerned with executing the PES and measuring its execution time. To perform this analysis, a tool was developed that captures the execution times of PES to allow analyzing its execution from different aspects. For instance, it provides information regarding thread's start and end times, scheduling of threads, the CPU time taken by each thread, end-to-end time of each thread and the program itself. The tool works, as depicted in Fig. 1, by reading the execution traces and detecting the starting and ending points of thread execution. An algorithm was developed to automate the process of calculating the thread execution time from the obtained traces.

To visualize the program flow based on its execution trace information, the capabilities of some existing tools were investigated. Initially, bar graph was used in MS-Excel to visualize the traces information. But, the data had to be changed in Excel worksheets and graph redrawn each time to generate a visualization. The traces information was very large for Tableau software tool1 to plot, resulting in scroll and zoom problems. Similarly, the Dot graph description language has the problem of scaling during modeling. This investigation led to build our own tool for visualization of executing threads. Consequently, the developed tool also provides a graphical visualization of the thread execution per core (Fig. 1), where each horizontal bar represents an executing user threads. These threads are shown in different colors along with their IDs and time stamps.

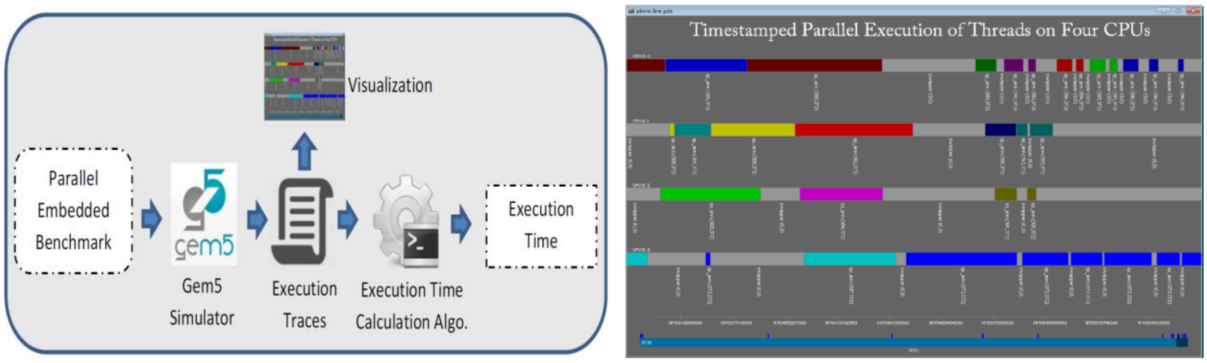
Visualization
Parallel Embedded Benchmark
gem5
Gem5 Simulator
Execution Traces
Execution Time Calculation Algo.
Execution Time
Timestamped Parallel Execution of Threads on Four CPUs

Fig. 1. Internal working of the developed dynamic analysis tool (Left), along with thread-core-based visualization (Right) of the execution of a PES.

4. Experiences and Lessons Learned

Observations and suggestions. The main challenge in flow analysis was the manual analysis of the source code. This was a laborious and error prone activity. An ideal case would be to semi/fully automate this process. In dynamic analysis, the main challenge was Gem5 setup for traces. The details are available in a technical report[2] for interested readers. Thread execution times for different benchmarks, shown in Fig. 2, were measured using the developed tool. Although load balancing and task decomposition are considered in the suite, the variations in the execution times of threads were observed. This is due to the inconsistent behavior of the operating system (OS) scheduler, such as random arbitration of swapper thread (idle task shown in gray color in Fig. 1) between the execution of user threads, switching of a thread among different cores. The captured thread-execution patterns across different benchmarks did not show any consistency that can be used as a basis for semi-automated, or automated flow analysis. Considering the unexpected behavior of OS a real-time OS and time-predictable system software can be alternate. Alternatively, the focus can be changed to the end-to-end measurement, where the overall thread execution would be taken into consideration instead of the inconsistent individual thread execution across several runs. Consequently, the end-to-end execution time of PES was measured using the generated execution traces.

[1] http://www.tableau.com/
[2] http://bit.ly/1JqheNS

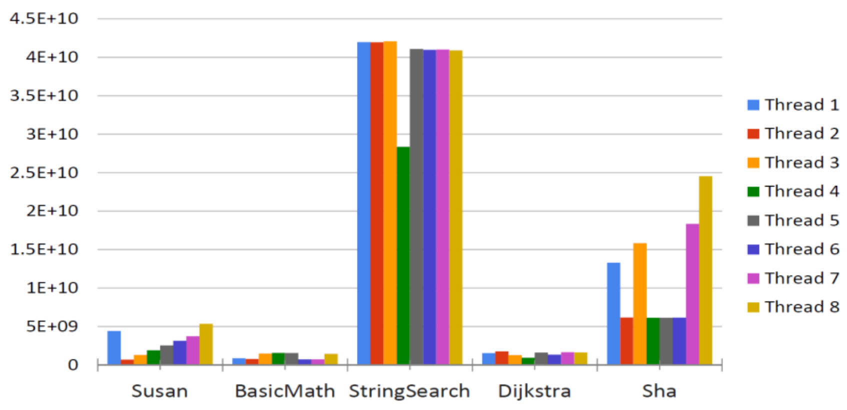

Fig. 2. Threads execution times for different benchmarks in number of processor clock cycles.

Typically, measurement-based approaches are not considered for safe estimates. Since the executions represent a subset of the actual possible executions, where a pathological worst-case could have been missed during testing. Hence, reliable guarantees of observing the worst case cannot be given. However, it is sufficient for soft embedded real-time software where occasional misses of deadlines are tolerated. But for safety-critical systems, where absolute safety of programs is required, static analysis should be performed. To guarantee safety, the measurement-based methods need to exercise a significantly large number of appropriate input values [4]. To heuristically find such inputs that will cause the program to execute for the longest period of time, a meta-heuristic optimizing search technique, such as a Genetic Algorithm (GA) can be utilized. To this end, the steps of GA were applied to automatically generate the inputs of PES that produce long execution times [5].

5. Related Work

In general, the research on WCET estimation on multi-core architectures has mainly focused on the predictability of accesses to shared resources. For example, the predictability features and the timing variance of single and multi-core processors are analyzed in [6]. Particularly, the research on high-level flow analysis of parallel applications running on multi-core processors is relatively new and very few studies are available to compare. The existing research on flow analysis assumes sequential execution of instructions, i.e., no concurrency is possible. In addition, the existing research is generally not mature enough to tackle parallel applications [7].

Rochange et al. [8] for the first time highlight the problem of analyzing timing behavior of non- sequential software on a multi-core architecture. Their manual analysis determines synchronization and communication between executing threads. However, the process described in the study is completely guided by the user and is specific to the estimation of WCET of one component of the parallel application (3D multigrid solver). Therefore, it needs to be investigated further before it can be generalized for other parallel applications. In [9], the classical control flow analysis is performed (separately) on the functions/tasks that are put in parallel, and then new edges are added between basic blocks to model synchronization/communications between tasks. One new edge is added for each inter-task communication along with duration to model the message transmission time for communications. However, instead of proposing a new technique, the work extends a WCET estimation method, where only the control flow analysis phase is modified.

Marref and Bernat [10], [11] have presented predicated WCET analysis, which deduces execution time effects between basic blocks from timing traces. These execution time effects are later injected into the calculation as constraints on execution paths, consequently tightening the WCET calculation. Although, there also exists a number of commercial and research prototypes static timing analysis tools (such as aiT, and SWEET), they are limited to sequential programs. Though some are extended to analyze parallel applications, such as Heptane and OTAWA [12], [13], but they are not open-source, e.g., Heptane extended version. Other tools are either in the experimental phases or have a different focus, e.g., multi-core Chronos is focused on hardware analysis.

6. Conclusion

This paper reports our experiences of performing the timing analysis of Parallel Embedded Software (PES). This includes a high-level flow analysis and low-level dynamic analysis. As a first step towards computing the WCET bounds, a high-level flow analysis of PES is performed in this work. Although the flow analysis of the source code was performed manually, focus was given to identify the kind of sub-analyses that can be automated, those that need user-provided annotations, and those that still need to be done manually. Besides providing useful suggestions to perform the timing analysis of PES, we derived WCET estimates of PES dynamically by using a tool developed to capture and visualize the execution times of threads, as well as the entire application on different cores. In future, the steps of the proposed flow analysis method can be fully automated, by developing a set of algorithms to identify dependencies and worst-case concurrency.

7. References

[1] R. Wilhelm, et.al. The worst-case execution-time problem – overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS), 7(3):36, 2008.

[2] N. Binkert, et al. The gem5 simulator. ACM SIGARCH Computer Architecture News, 39.2(2011): 1–7.

[3] S. M. Z. Iqbal, Y. Liang, and H. Grahn. Parmibench-an open-source benchmark for embedded multiprocessor systems. Computer Architecture Letters, 9(2):45–48, 20.

[4] A. Betts, G. Bernat, R. Kirner, P. Puschner, and I. Wenzel. WCET coverage for pipelines. RealTime Systems Research Group-University of York and Institute of Computer Engineering-Vienna University of Technology, Tech. Rep, 2006.

[5] M. W. Aziz and S. A. B. Shah. Test-data generation for testing parallel real-time systems. In Testing Software and Systems, pages 211–223. Springer, 2015.

[6] D. Kästner, et al. Meeting real-time requirements with multi-core processors. In Computer Safety, Reliability, and Security, pages 117–131. Springer, 2012.

[7] B. Lisper, et al. Practical experiences of applying source-level WCET flow analysis on industrial code. Leveraging Applications of Formal Methods, Verification, and Validation, 449–463, 2010.

[8] C. Rochange, et al. WCET analysis of a parallel 3D multigrid solver executed on the merasa multi-core. In International Workshop on Worst-Case Execution Time (WCET) Analysis, volume 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[9] D. Potop-Butucaru and I. Puaut. Integrated worst-case response time evaluation of multicore non-preemptive applications. In Research Report RR-8234. INRIA, 2013.

[10] A. Marref and A. Betts. Accurate measurement-based wcet analysis in the absence of source and binary code. In Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on, pages 127–135. IEEE, 2011.

[11] A. Marref. Predicated worst-case execution-time analysis. Ph.D. dissertation, York, UK, 2009.

[12] D. Potop-Butucaru and I. Puaut. Integrated worst-case execution time estimation of multicore applications.In Workshop on Worst-Case Execution Time Analysis, pages 21–31, 2013.

[13] H. Ozaktas, C. Rochange, and P. Sainrat. Automatic wcet analysis of real-time parallel applications. In International Workshop on Worst-Case Execution Time (WCET) Analysis, volume 30. Schloss Dagstuhl, 2013.