Survey : Various Methods for WCET Estimate Calculation

Manjiri K. Kulkarni
Department of Computer Engineering
Pimpri Chinchwad College of Engineering
Pune-44

J.S. Umale, PhD
Professor, Department of computer Engineering,
Pimpri Chinchwad Collage of Engineering
Pune-44

ABSTRACT
The design of a real-time system revolves heavily around a model known as a task schedule, which allots computational resources to executing tasks, i.e. programs. Many different scheduling algorithms have been invented, all of which depend on a set of temporal properties relevant to each task. One such property is the Worst Case Execution Time (WCET), intuitively described as the longest possible execution time. It is required to determine variation in execution times. If the variation is bounded then the system has time predictable behavior. Otherwise, we cannot provide any guaranties for the worst case execution time and the architecture is time unpredictable. Embedded controllers are expected to finish their tasks reliably within time bounds. Task scheduling must be performed essential: upper bound on the execution times of all tasks statically known Commonly called the Worst-Case Execution Time (WCET). To use the GPUs in real time systems it is required to have time predictable behavior. However, it is hard to give an estimation of the WCET of a GPU program.

In this paper , we focused on comparative analysis of various WCET estimate techniques with their results evaluations as well as observations.

Keywords
WCET, IPG, ETP, Static Analysis, Hybrid Analysis, GPU.

1. INTRODUCTION
An embedded system like a Real-Time System (RTS) for which some special operation depends on timing constraints. The design of a real-time system closely related to the task schedule model, which allots the CPU resource to executing tasks, assuming access to maximum time required that means the Worst-Case Execution Time (WCET) of each task. However, determining the actual WCET is not easy because software and hardware properties both because variation in execution times. In WCET estimates, the main thing is to bound the actual WCET so that the task schedule is not compromised. Techniques for WCET analysis are as follows :

1. End-to-End : The High Water-Mark Time (HWMT) is the end-to-end longest observed execution time which lies in close proximity to the actual WCET[16].
2. Static Analysis (SA) : Static analysis have two different models: Small segments of the software that means program model and the functional and temporal behavior given by processor model of the hardware. By combining both, resulting in a WCET estimate[16].
3. Hybrid Measurement-Based Analysis (HMBA) : It collects the execution times of program segments via instrumentation points (ipoints)[16].

Nowadays, Graphic Processing Units (GPU) have drawn increasing popularity for high performance computing. The NVIDIA Compute Unified Device Architecture (CUDA) summarizes GPU as a general-purpose multithreaded SIMD (single instruction, multiple data) architectural model, and provides a C-like interface supported by a compiler and a runtime system for GPU programming. As in the case of CPU programming, ensuring that a GPU application efficiently utilises computational resources is a cardinal goal. Mostly it involves analysing average case performance and optimising accordingly, but outlier execution times, such as the WCET, also prove fruitful.

2. LITERATURE SURVEY
1. In the work, ***WCET Analysis of Probabilistic Hard Real-Time System (2002)*** [5], the evaluation of WCET estimate is on the basis of probabilistic analysis, in which the notion of probabilistic hard real-time system which has to fulfill all the deadlines but for which a probabilistic (high) guarantee requirements are introduced. They also combine both analytical and measurement approaches into a model, for computing probabilistically bounds on the execution time of the worst case path of sections of code. In this work, technique presented is based on combining (probabilistically) individual’s worst case effects blocks to build the execution time model of the program’s worst case path but in case of may have not been observed in the measurements. Here focus is on a particular use of Execution Profiles in the domain of WCET analysis [9]. The “events” frequencies are represented by the execution profiles are the different execution times that a piece of code which may require to execute. The relative frequencies which are represented by such an execution profiles of execution times is an execution time profile (ETP) where The EPs could also be provided by analytical methods as the ones used in static WCET analysis[6][3]. To find the longest execution time of a program is by combining together the observed execution time of its parts. This combination should be go towards the worst case.

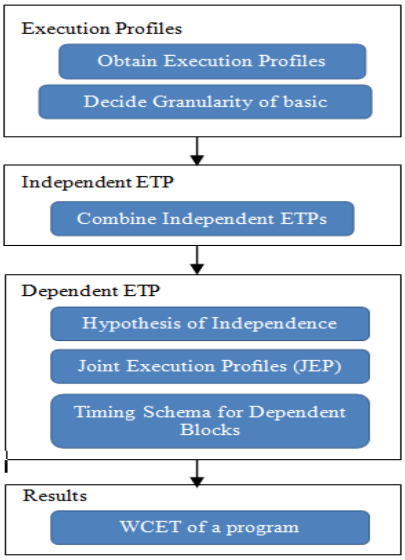

Fig. Probabilistic WCET Analysis

The above figure shows the WCET of program on the basis of execution profiles, in the first step, obtain Execution profiles of code by defining granularity. In the second step, independent ETPs are identified and combine all those identified ETPs in the next step dependent ETPs are identified and assuming independent execution of sections of code may be pessimistic or optimistic.If there is strong positive correlation is present between the execution times of certain pairs of execution blocks by taking optimism and pessimism of the hypothesis of independence then it is an optimistic estimate.Then apply timing schema according to the dependency information available.

2. In the work, *Deriving the Worst-Case Execution Time Input Values (2009)* [12] based on a combination of input sensitive static WCET analysis [10] and systematic search over the value space of the input variables, to derive the input value combination that causes the WCET unlike in previous work where probabilistically WCET estimate was derived. There are present several different approaches to speed up the search and evaluations which show that, for many type of programs, the WCET input values can be relatively quickly derived, even for program with large input value spaces. It show that the from WCET input values derived WCET estimates often are muc1h tighter than the WCET estimates derived when combinations of all possible input value are taken into account. A novel search algorithm based on a combination of static WCET analysis and systematic search over the input variables's value space are used to find the WCET input values. Many static WCET analyses are input-sensitive, meaning that when calculating a WCET estimate, they are able to take constraints on input variable values into account. When static input-sensitive WCET analysis tool run with a single worst-case input value combination, it will be able to produce a tighter WCET estimate, as compared to when it is run with all possible input value combinations. This allows for better utilization of overall system and for the real-time system designer makes it easier to produce a schedulable system.

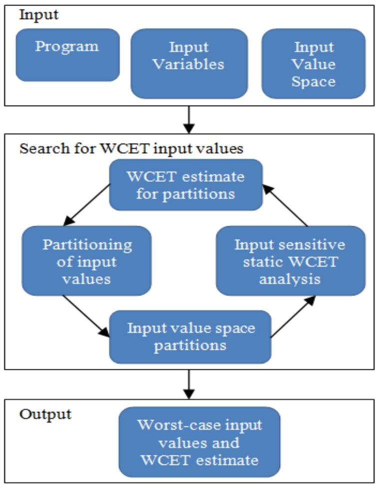

Fig. WCET input value analysis

This figure shows that the flow of WCET input value analysis [12], in which first block is of input which contains program, input variables and input value space as input the in second block the search algorithm systematically divides this input value space into smaller partitions of input value space, each with a subset of the input value space. Algorithm will works iteratively by calculating WCET estimates for different partitions input value space of the program. The largest WCET estimate is selected from each iteration of partition and divided into two or more smaller partitions, for which WCET calculations are made. The process continues until the selected partition holds only one input value combination, which is then returned. In the last block, worst case input values and WCET estimates are calculated.

3. In the work, *WCET Analysis of Component-Based Systems using Timing Traces (2011)* [14] shows how to obtain a safer WCET estimate of a Real Time Systems which are composed of components using time-stamped traces of program execution . For this, data like program model, execution times, execution bounds are needed in the WCET computation, which is derived from parsing traces. The trace-parsing stage produces the structure of the *Instrumentation Point Graph (IPG)* [4] and derives the execution times and execution bounds of its edges: the calculation engine is then tasked with producing a WCET estimate from these data. Here *Implicit Path Enumeration Technique (IPET)* is used which is basically maximised an objective function and it subject to a number of constraints since it can easily model arbitrary control flow and is not therefore hindered by the irreducibility of an IPG [1].

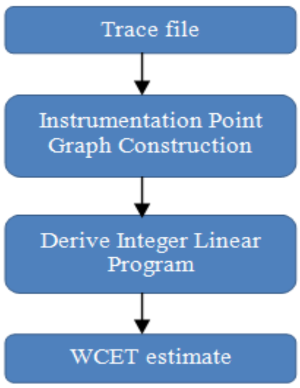

Fig. WCET estimate from IPG

In the above figure, WCET estimate is calculated from instrumentation point graph. In the first step, IPG is constructed from *trace file* (set of timing traces). In the second step, Integer Linear Program is derived form IPG. In above model, upper capacity constraints are execution bounds derived from trace parsing. Solving this model via standard (integer) linear program solvers returns both a WCET estimate and a setting of the execution count for each IPG edge in the worst case. In this way, all paths are implicitly considered since the solver attempts different assignments to the execution counts in determining the worst case. When the execution times and upper capacity constraints on the decision variables are safe, the solution to the ILP always returns an upper bound on the actual WCET [7].

4. In the work, *Estimating the WCET of GPU-Accelerated Applications using Hybrid Analysis (2013)* [15] is proposed to enhance optimizations in GPU programming languages such as CUDA, OpenCL, requires optimization which is highly depends on workload and structure of input data in parallelism and locality by minimizing synchronization. In this, from traces of execution, execution times of small program segments are deduced and a calculation backend derived from the Control Flow Graph (CFG) produces a WCET estimate.

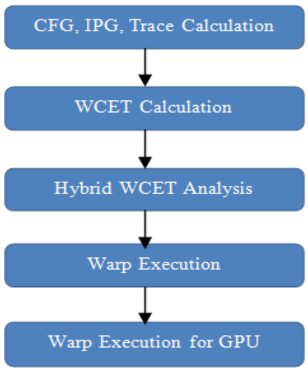

Fig. WCET Estimate for GPU-accelerated applications

In the above figure, first step is of obtaining *Contol flow Graph(CFG), Instrumentation Point Graph (IPG),*traces of all possible execution paths. In second step, IPG transformed into a tree representation that is similar to an abstract syntax tree in which its internal vertices represent sequential, alternative, and iterative constructs, while leaves represent Ipoint transitions as these are the atomic units of computation [15]. Individual threads within a thread-block execute on a specific core. However, on an NVIDIA GPU, threads are not the atomic unit of scheduling, it is a sub-group of the thread-block called a *warp*. The number of threads in a warp, i.e. the warp size, has remained 32 across all NVIDIA GPUs. When different threads in a warp want to follow different sides of an if-then-else construct, branch divergence occurs. For each path specific warp is allocated so that each path can be execute parallelly and performance will get increased. For GPU-accelerated applications, this type of dynamic execution method is suitable.

3. OBSERVATIONS

In the first work, there is probabilistic model which uses static analysis of execution profiles but all execution paths does not get explored so worst case time required for execution is not exact and it is executed on CPU.

Likewise, in second work towards WCET estimate, it also uses static anslysis but in this work, search algorithm is introduced which searches worst case execution time. structure of the input plays main role in opimization therefore in input sensitive specified application it is useful but no parallelism.

In the third work, it gives WCET for real time system on the basis of time traces and it shows that how to obtain a safer WCET estimate of a Real Time Systems which are composed of components using program execution's time stamped traces from IPG by using IPET but it cannot determine the exact longest path from the execution counts because the order of execution is missing. Also no parallel execution so not suitable for GPU based applications.

In the fourth work, more improvement in the WCET estimation accuracy as well as in performance. In this hybrid model comparative to all previous model, it works better because at front end it reduces execution time by reducing traces and at backend CFG will generated and WCET estimate calculated.

4. RESULTS AND INTERPRETATIONS

In the first work (2002), the results in the case studies show[5], that a partially known dependencies between sections of code, enhance the properties of the resulting execution profile of a program considerably.

In the second work (2009), the number of WCET calculations needed grow with the size of the input value space. All programs experience varying WCET calculation time. In general, when the input value size decreases, the time for performing the WCET calculation also decreases. Thus, the first analysis generally consumes most time, while subsequent analyses are faster. *OrgW* gives original WCET estimate (in clockcycles) derived by SWEET Tool with all input value combinations and *FinW* gives the final WCET estimate obtained for the derived worst-case input value combination [12].

Table 1

Program	LOC	OrgW	FinW
Cyclic redundancy check computation.	128	83275	83275
Nested loop program	64	4557	528
Search in a multi-dimensional array	535	1603	1603

In the third work(2011), as shown in Table 2 [14] the benchmarks under investigation are taken from the Ma¨lardalen suite [8], which are used by many groups in WCET analysis to evaluate their tools. In this evaluation they are particularly appealing since the worst-case TVs(Test Vectors) are easy to deduce [5].

Table 2

Application	HWMT	Actual WCET	Basic Block	Branch	Pre-dominator
bubblesort	1008	1028	1818	1818	2310
expint	10721	10956	16644	16644	12414
insertsort	1113	1175	1799	1799	2124

In forth WCET estimate work(2013), analysis of CUDA applications shipped with the CUDA SDK [5] and selected those for which the application performs meaningful computation (some benchmarks merely illustrate a CUDA feature) and for which it was straightforward to generate a test vector. The specific benchmarks analysed are given in Table 3 [15].

Table 3

Applications	HWMT	Warp WCET	Hybrid WCET
BitonicSort	1,045,259	173,548	6,929,575
EigenValues	1,143,429	2,801,330	2,801,337
MatrixMultiply	3,642	4,678	4680

5. CONCLUSIONS

In this survey work, comparative analysis of various methods to calculate WCET estimate are studied and from this comparative analysis, it is conclude that WCET estimate is calculated according to application to be executed. But for GPU accelerated applications, hybrid WCET model is best and it is observed that before going for large segment code optimization, it is always better to go for small segment optimization first.

6. FUTURE WORK

In future, it can be possible that to automatically diagnose performance bottlenecks in GPU applications using hybrid performance model, and it can be useful to increase the performance of WCET estimate.

7. REFERENCES

[1] P. Puschner and A. Schedl, "Computing Maximum Task Execution Times - A Graph-Based Approach," Real-Time Systems, vol. 13, no. 1, pp. 67–91, 1997.

[2] J. Wegener and M. Grochtmann, "Verifying timing constraints of realtime systems by means of evolutionary testing," Real-Time Systems, vol. 15, no. 3, pp. 275–298, 1998.

[3] A. Colin and I. Puaut. A modular and retargetable framework for tree-based WCET analysis. In Proceedings of the 13th Euromicro Conference on Real-Time Systems, pages 37–44, Delft, Netherlands, June 13–15 2001.

[4] J. L. Diaz, D. F. Garcia, K. Kim, C. Lee, L. Lo Bello, J. M. Lopez, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In Proceedings of the 23rd Real Time Systems Symposium RTSS 2002, Austin, Texas, USA, Dec. 3–5 2002.

[5] Guillem Bernat Antoine Colin Stefan M. Petters, "WCET Analysis of Probabilistic Hard Real-Time Systems", Real-Time Systems Research Group Department of Computer Science University of York, UK, 2002.

[6] R. Arnold, F. Mu¨ller, D. Whalley, and M. Harmon. Bounding worst–case instruction cache performance. In Proc. of the IEEE Real–Time Systems Symposium (RTSS'94). IEEE Computer Society Press, Dec. 1994. [3] I. Berger. Can you trust your car? IEEE Spectrum, 39:40–45, Apr. 2002.

[7] A. Colin and S. M. Petters, "Experimental Evaluation of Code Properties for WCET Analysis," in RTSS, 2003.

[8] A. Aho, R. Sethi, M. S. Lam, and J. Ullman, Compilers: Principles, Techniques and Tools, 2nd ed. Addison-Wesley, 2006.

[9] A. Betts, "Hybrid Measurement-Based WCET Analysis using Instrumentation Point Graphs," Ph.D. dissertation, University of York, November 2008.

[10] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenstr¨om. The worst-case execution time problem — overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS), 7(3):1–53, 2008.

[11] 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. Stenstro¨m, "The Worst-Case Execution-Time Problem–Overview of Methods and Survey of Tools," ACM Transations on Embedded Computing Systems, vol. 7, no. 3, pp. 1–53, 2008.

[12] Andreas Ermedahl, Johan Fredriksson, Jan Gustafsson, "Deriving the Worst-Case Execution Time Input Values", 21st Euromicro Conference on Real-Time Systems, 2009.

[13] Ma¨lardalen University WCET project homepage, http://www.mrtc.mdh.se/projects/wcet, May 2010.

[14] Adam Betts, Amine Marref, "WCET Analysis of Component-Based Systems using Timing Traces", 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011.

[15] Adam Betts and Alastair Donaldson, "Estimating the WCET of GPU-Accelerated Applications using Hybrid Analysis", 25th Euromicro Conference on Real-Time Systems, 2013.