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 guarantees 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 utilizes computational resources is a cardinal goal. Mostly it involves analyzing 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.
The above figure shows the WCET of program on the basis of execution profiles, in the first step, obtain execution profile of code by defining granularity. In the second step, independent ETPs are identified and combine all those identified ETPs in the next step dependent ETs 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 much 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.

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].
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 (2015) [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.

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>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>OrgWC</th>
<th>FinWC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclic redundancy check</td>
<td>128</td>
<td>83257</td>
<td>83257</td>
</tr>
<tr>
<td>Nested loops</td>
<td>64</td>
<td>4537</td>
<td>528</td>
</tr>
<tr>
<td>Search in a multi-dimensional array</td>
<td>523</td>
<td>1003</td>
<td>1003</td>
</tr>
</tbody>
</table>

In the third work (2011), as shown in Table 2 [14] the benchmarks under investigation are taken from the Ma’tardalen 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>
<thead>
<tr>
<th>Application</th>
<th>HWMT</th>
<th>Actual WCET</th>
<th>Basic Block</th>
<th>Branch</th>
<th>Predominant</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubblesort</td>
<td>1008</td>
<td>1028</td>
<td>1818</td>
<td>1818</td>
<td>2510</td>
</tr>
<tr>
<td>expnlist</td>
<td>1072</td>
<td>10956</td>
<td>16644</td>
<td>16644</td>
<td>12414</td>
</tr>
<tr>
<td>insertsort</td>
<td>1113</td>
<td>1175</td>
<td>1799</td>
<td>1799</td>
<td>2124</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Applications</th>
<th>HWMT WCET</th>
<th>Warp WCET</th>
<th>Hybrid WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitonicSort</td>
<td>1,045,259</td>
<td>173,548</td>
<td>6,929,575</td>
</tr>
<tr>
<td>EigenValues</td>
<td>1,143,429</td>
<td>2,801,330</td>
<td>2,801,337</td>
</tr>
<tr>
<td>MatrixMultiply</td>
<td>3,642</td>
<td>4,678</td>
<td>4680</td>
</tr>
</tbody>
</table>

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


