EXTENDING THE PATH ANALYSIS TECHNIQUE TO OBTAIN A SOFT WCET

Paul Keim, Amanda Noyes, Andrew Ferguson
Joshua Neal, Christopher Healy[1]

Abstract

This paper discusses an efficient approach to statically compute a WCET that is "soft" rather than "hard". The goal of most timing analysis is to determine a guaranteed WCET; however this execution time may be far above the actual distribution of observed execution times. A WCET estimate that bounds the execution time 99% of the time may be more useful for a designer in a soft real-time environment. This paper discusses an approach to measure the execution time distribution by a hardware simulator, and a path-based timing analysis approach to derive a static estimation of this same distribution. The technique can find a soft WCET for loops having any number of paths.

1. Introduction

In order to properly schedule tasks in a real-time or embedded system, an accurate estimate of each task's worst-case execution time (WCET) is required. To fulfill this need, a static timing analyzer is used to estimate the WCET. This WCET is guaranteed to be greater than or equal to the actual execution time no matter what the input data may be. However, in some cases this predicted WCET may be quite loose, because the probability of taking the worst-case path all the time may be minuscule.

There is a distinction between a hard and a soft real-time system. In a hard real-time system, deadlines are absolute, and thus the WCET estimate must never be less than the actual execution time. Timing analysis is typically concerned with this case of finding this "hard WCET." By contrast, in a soft-real time system, such as media production, an occasional deadline may be tolerated. This paper is concerned with the notion of a "soft WCET" – a WCET estimate that is "almost" always above the actual execution time, one that is guaranteed to be correct at least, say, 99% of the time.

It has been customary for WCET analysis to address hard real-time constraints. However, it could be the case that the WCET is much higher than the actual execution times experiences. Even the 99th percentile of the execution time distribution could be much lower than the computed WCET. And if this is the case, the WCET bound would be overly conservative in a soft real-time environment, in which an occasional missed deadline could be tolerated.

[1] Department of Computer Science, Furman University, Greenville SC 29613 USA; e-mail: chris.healy@furman.edu

The main contribution of this research is to determine an efficient means to compute a soft WCET, and we do so by first determining the whole distribution of execution times. Probability distributions are continuous functions, so to make our task manageable, we quantize this distribution by creating an array of buckets, in which each bucket represents one possible execution time. In practice, we selected an array size of 100 buckets, so that we in effect store percentiles of the execution time distribution. The last bucket, containing the highest value, will contain the "soft WCET". Analogously, the first bucket would contain the lowest value or "soft BCET", but our study was focused only on WCET.

2. Related work

This paper is inspired by the work of other researchers to apply statistics and probability techniques to timing analysis and real-time systems. Specifically, the pioneering work of Bernat et al. in 2002 proposed a stochastic or probabilistic approach to WCET analysis [2]. Stochastic analysis means we want to look at the overall distribution of execution times. Then, we can observe where the WCET lies in relation to the bulk of this distribution. Another statistical approach was proposed by Edgar [4], in which he uses the Gumbel distribution to infer the WCET from a sample of data points given that the distribution of execution times is unknown in advance. Atlas and Bestavros even used similar statistical techniques in order to generalize and relax the classical Rate Monotonic Scheduling algorithm [1]. This paper differs from other recent approaches mainly in that we are concerned with creating a data structure that can capture the overall distribution of execution times, rather than focusing on the extreme tail of the distribution.

3. A Simple Example

Figure 1 shows a simple scenario. Suppose we want to compare the WCET with the overall execution time distribution for the following simple loop:

```
for (i = 0; i < 100; ++i)
  if (a[i] > 0)
    ++count;
```

After compiling, the `if`-statement becomes a conditional branch that can either be taken or not. Thus, this loop has two paths, A and B. The probability of taking each path is p_A and p_B respectively. We also need to know *n*, the number of loop iterations, which is 100 in this example. We can now easily compute the probability of taking path A *k* times out of *n* iterations using the binomial probability formula:

$$prob = \frac{n!}{k!(n-k)!}\ p_A^k p_B^{n-k}$$

If we know nothing about the values in the array a, we assume there is an equal chance of taking either path. Static timing analysis can tell us that the execution times of these two paths are 6 and 12 cycles. Therefore the execution time for the loop is between 600 and 1200, and we obtain the results in Figure 1. Had the probabilities been unequal, the overall shapes would be almost

unchanged: they would merely be shifted to the left or right and skewed (i.e. asymmetric) to a small degree. The extreme execution times at the tails would still practically never be attained.

Note that if a loop has only one path, it would not have an interesting distribution. Both the probability and execution time distribution graphs would have a single spike with a height of 1.00. If a loop has more than two paths, then there is the potential for multiple peaks in the distributions.

The point to take away from this example is that the WCET, which is about 1200 cycles, is very unlikely to be achieved. There is more than a 99% chance that the actual execution time will be less than 1000 cycles. No amount of loop analysis can tighten the hard WCET estimate without further knowledge of the distribution of values of the array. On the other hand, a soft WCET calculation can bound the execution time distribution much more tightly.

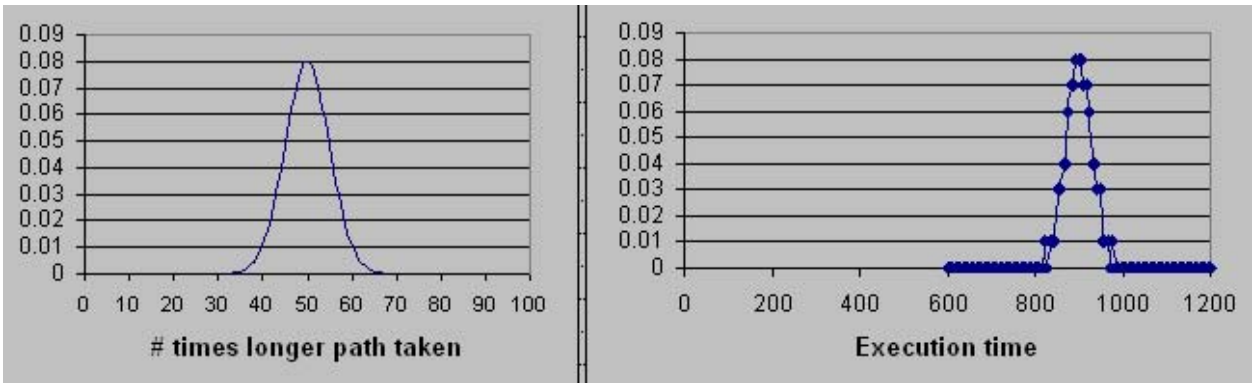

Figure 1: Probability and execution time distributions.

4. Overview of the Approach

Our approach utilizes two separate tools: a hardware simulator and timing analyzer. Details on the background of our approach can be found elsewhere [5]. The purpose of the simulator is to check the timing analyzer's predictions for accuracy. We use a simulator so that it is straightforward to make changes to the hardware specification, and it is straightforward to pinpoint sub-execution times for portions of the program, if necessary. We adapted the simulator to run a benchmark executable with randomized input values. After a set of trials, it then creates a histogram of the execution time distribution. The right tail of this distribution is the observed WCET of the trials.

Our methodology for running the simulator was straightforward: we would run each benchmark 1000 times with randomized input data. To make things more interesting we considered "fair" versus "unfair" distributions of input data. By "fair" we mean that the input data was generated in such a way that the paths would logically be taken about an equal number of times. For example, if there is a single if-statement checking to see if an array element is positive, then we would arrange a "fair" test to ensure that the array elements had a 50% chance of being positive. To generate "unfair" distributions, we made one path 3 times as likely to be executed.

The second tool is the timing analyzer that we used is an adaptation of an earlier version that computes hard WCET's. It relies on control-flow information determined from a compiler. Then, it represents the input program in a tree that hierarchically organizes the constituent functions, loops, basic blocks and instructions. Each loop's path information is then created and also stored with the

appropriate loop. The evaluation of the WCET proceeds in a bottom-up fashion, starting at the lowest level of detail (the instruction) and eventually working its way to the `main()` function. One important part of the timing analysis approach is the loop analysis which must examine the execution paths.

In working with both the hardware simulator (which measures execution time dynamically) and the timing analyzer (which estimates execution time statically), our goal is that the soft WCET predicted by the timing analyzer should be close to the longest execution time observed by the simulation.

5. Loop Analysis Algorithm

How do we statically generate a distribution of execution times? To illustrate our approach, let us assume that we have a loop with *n* iterations. If the loop has only one execution path, then this case is not interesting, since the execution time will not depend on the input data. In this trivial case, the execution time distribution would be completely flat and uniform. There will only be a meaningful distribution of execution times if the loop has two or more paths.

A loop's execution time array is created. In practice we chose our execution time array to hold 100 values, but this is simply an input parameter to the algorithm and can easily be changed. In our case, each cell represents 1 percentile of the distribution of execution times for the loop in question. We have found that this is sufficiently granular to capture a nearly continuous distribution.

The following algorithm illustrates the case of a loop having exactly two paths.

Objective: to estimate the execution time distribtion of a loop
Let n be the number of iterations.
First, find execution time of each path, which is assumed to be a constant number of cycles.
Second, assign probabilities to each path. Infeasible paths are given a probability of 0.0 and not included. Otherwise, we assume each path is equally likely to be taken.
Of the 2 paths, let's call A the longer path, and B the shorter.
previous_cumulative_prob = 0
For i = 0 to n:
    prob = binomial probability that path A executes i times and B executes n-i times.
    cumulative_prob += prob
    for each integer j between 100*previous_cumulative_prob and 100*cumulative_prob
        time_dist[j] = time taking A i times + B (n-i) times.

If a loop has three or more paths, there are two possible approaches. One approach is to repeatedly apply our binomial probabilities on each conditional branch we encounter. Otherwise, we can attempt to assign every path a separate probability and apply a multinomial probability distribution. Formulas involving multinomial probabilities can become quite complex [9], so we decided on the former approach.

As a simple illustration, here is a motivation for how we would handle 3 paths using the binomial probability method. Our three paths are called p1, p2, and p3. Paths p1 and p2 have a 25% each of being chosen, while p3 has a 50% chance of being chosen. What we can do is take the distribution

graph of p1 and p2, generated through binomial distribution of an equal probability using the 2-path algorithm above (both are 25%, but since they're the only ones we're allowing to run, it's actually 50%, a coin flip). Then, with the distribution {p1, p2}, we combine this with {p3}. Both now have a 50% probability, so we take the 100 values of each array, merge them into an array having 200 values, sort them in ascending order, and then *remove every other element* so that we end up with 100 values again, and this becomes our distribution array for all three paths.

Here is the algorithm for the general case of 3 or more paths in a loop:

Let $\oplus$ represent the application of the earlier binomial probability formula. We also need to weight the distribution if the probabilities of the paths are different in the following manner:

a. Weight the number of elements. Suppose the probability associated with each path is such that P2>P1. Thus, the 2nd path's array continues to contain 100 elements, and the combined array will stretch to include (100*P2/P1) elements.

b. To create the extra elements for the combined distribution, starting with the initial 100 elements, duplicate accordingly. If needing 200 elements, duplicate each element. If needing 150 elements, duplicate every other element.

Let $\{P_1,P_2,\ldots P_n\}$ be the set of paths, each with the probability of being executed P. We create the probability distribution D as follows.

1. **Use the binomial probability on $\{P_1,P_2\}$ to initialize D.**
2. **For i = 3 to n, D = D $\oplus$ P_i**

This elegant solution scales to any number of paths, because it is a repeated application of the binomial probability formula, rather than using a much more complex multinomial one.

6. Results

In our methodology, we assume that the loop under consideration reads from up to two globally declared arrays. These arrays may be multi-dimensional, and may contain either integers or floating-point values. We created 1000 versions of the benchmark, each with a different set of values for initializing the array(s). The initialization is done at the global declaration so that it does not impact the execution time of the benchmark under review. The benchmarks we examined were the same as those used in previous work.

One could note that there is a potential for some lack of precision. There is always a chance that running a program 1000 times will miss the absolute worst case input data. However, our results show that in most cases, the distribution of execution times is relatively tight. This can happen for two reasons. First, the conditional-control flow that depends on the program's input data can be drowned out by the rest of the program that does not depend on the input data. Second, it is often the case that when there are two paths, their execution times do not differ a great deal.

Actually, we found that testing our timing analyzer against the simulator was a little easier to automate than in our previous work with "hard" WCET's. Before, we would compare the static prediction of the WCET with a single measurement. That measurement used a specific input which we wrote by hand, studying it to make sure it is the worst-case input data. However, in general this is not always feasible. This is another reason why we decided to adopt a stochastic approach.

There may be benchmarks for which it may be very difficult to determine the input data that will drive every worst-case path throughout the program. But with a stochastic approach, you can observe the probability distribution and focus your attention on the highest bucket (e.g. the 99th percentile).

Table 1 shows some of the benchmarks we ran, and how they are characterized based on their observed stochastic behavior. These benchmarks have been used in previous work [6]. The numbers in parentheses indicate the execution time of the longest trial of that benchmark, expressed as a percentage of its statically predicted absolute WCET. If this value is much below 100, this does not indicate that the static WCET prediction is inaccurate. Rather, it indicates that the actual WCET, i.e. taking every opportunity to execute a worst-case path, is very rarely encountered (and indeed, was not encountered in the 1000 trials). Note that in the case of the very narrow distributions, the skewness of the distribution is insignificant.

What is interesting about these results is that the same benchmark with different input data (fair versus unfair) can have a markedly different distribution of execution times. For example, in the case of the Sym benchmark, which tests a matrix to see if it is symmetric, a fair distribution yields a simulated execution time that was never more than 2% of the hard WCET. But our unfair distribution that made one path 3 times as likely to be taken brought up the maximum observed execution time to 19% of the hard WCET.

	Width 0-2%	Width 3-25%	Width >= 26%
Skewed +	Sumnegallpos-fair (98) Summatrix-fair (93) Sym-fair (2)	Sumnegallpos-unfair (100) Sym-unfair (19)	Sprsin-unfair (98)
Skewed -	Neville (80)	Summatrix-unfair (100)	Bubblesort-fair (85)
No skew	Unweight-fair (95)	Sprsin-fair (74) Unweight-unfair (100)	Bubblesort-unfair (86) Sumoddeven (79)

Table 1: Stochastic characteristics of selected benchmark programs.

Additionally, the number of paths affects the distribution. We were primarily interested in benchmarks with 2 paths. Generally this means we could expect one peak in the distribution, and we could use the theory of binomial probability distribution to predict their behavior. However, it is interesting to note that even benchmarks with many execution paths still had just one observable peak in the distribution. Other peaks could exist, but they may be relatively insignficant.

The analysis of a single program can become more interesting when you combine the effect of more than one loop. For example, the `Sprsin` benchmark has three loops, each with a different execution time distribution classification.

- Loop 1's distribution is uniformly around 100%.
- Loop 2's distribution is tightly between 77 and 83%.
- Loop 3's distribution varies widely between 67 and 90%.

The overall behavior of sprsin is similar to loop 2.

To illustrate the robustness of the loop analysis algorithm of Section 5 to a non-trivial number of paths, we show an example benchmark containing a loop with 7 paths. Its source code is given below. Figure 2 shows the two execution time distributions: the 1000 trials from the simulator, compared against the histogram of the 100 buckets created by the timing analyzer.

```
int a[60][60], pos, neg, zero;

main() {
  int i;

  for (i = 0; i < 60; ++i) {
    if (a[i][0] > 0 && a[0][i] > 0)
      ++pos;
    else if (a[i][0] == 0 && a[0][i] == 0)
      ++zero;
    else
      ++neg;
  }
}
```

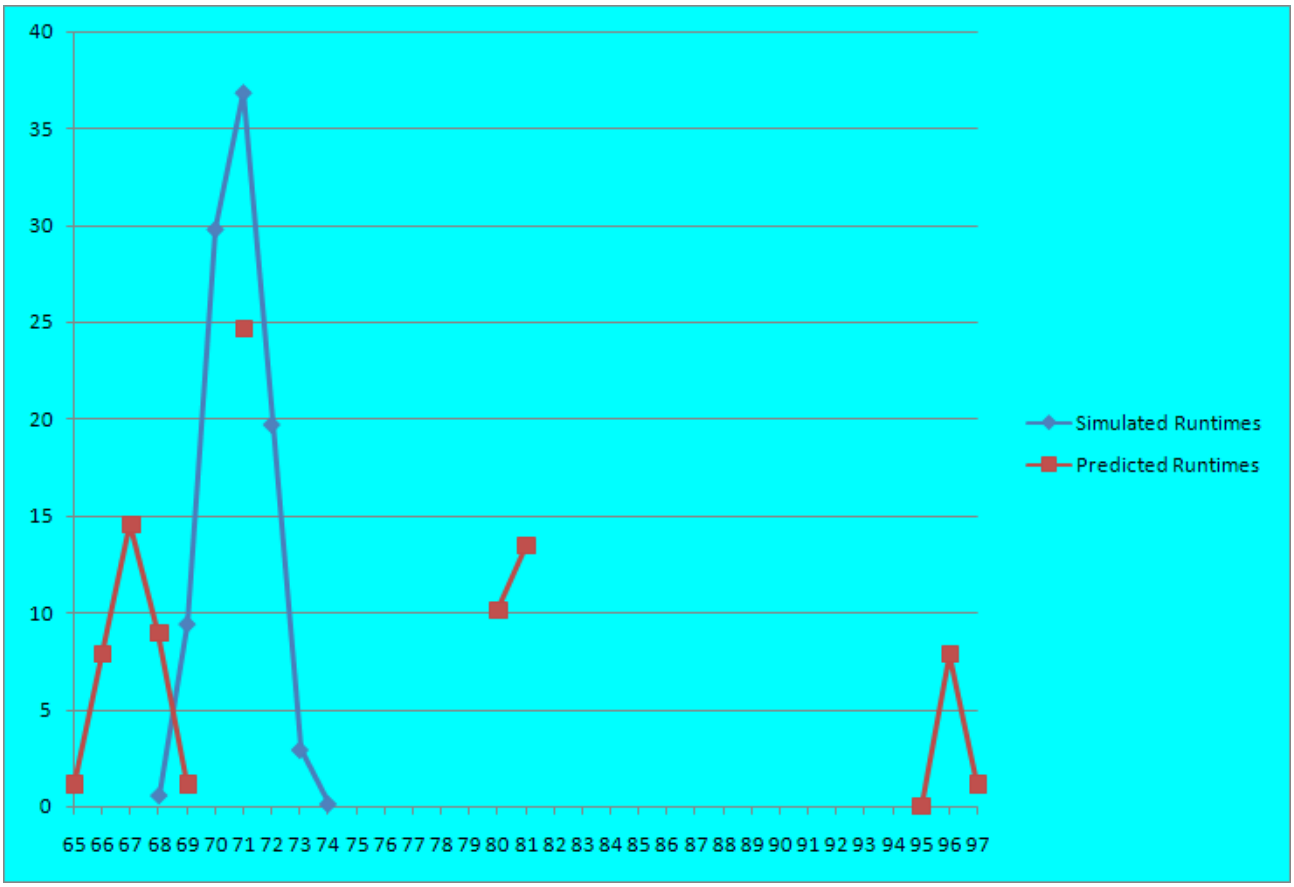

Figure 2: Percentage histogram of the simulated and predicted runtimes for the 7-path benchmark example. The horizontal axis refers to the percentage of the hard WCET, and the vertical axis is the percentage of trials.

The observed execution times never exceeded 74% of the hard WCET. Meanwhile the soft WCET predicted by the timing analyzer was 97% of the hard WCET. This example illustrates a current limitation to our approach: we assume each conditional branch taken 50% of time, which is not always realistic. This accounts for much of the discrepancy. The timing analyzer can do a better job if it has some knowledge of the probability of a certain branch being taken.

7. Future Work

Currently, our approach works for single loops containing any number of paths. So, naturally, the next step would be to handle multiple loops in a program, whether nested or consecutive. For example, we could make use of the convolution technique described by Bernat et al. [2] to compose multiple execution time distributions. In order to make the timing analyzer's execution time distribution (and its soft WCET) more accurate, we also need to specifically analyze branch instruction. For example, loop branches are usually taken, and we can at least use a better heuristic in such a case.

Another direction of future work is a refinement of how we handle probability. Currently, we treat iterations independently, and the choice of path on each iteration is random. In the future, we can take „runs" into account [8], *i.e.* the fact that the same path may be taken many times in succession. In particular, if the same (long) path is taken for many consecutive iterations, this will temporarily bias the execution time distribution upward, closer to the absolute WCET. For example, in a loop with 1000 iterations, it may be interesting to examine the distribution of execution times of any set of 50 consecutive iterations of the loop.

In addition, we can also combine our stochastic approach with our recent work on parametric timing analysis [3]. Currently, our approach here assumes that the execution times are constant values and that the number of loop iterations is known at compile time.

8. Conclusion

A timing analyzer's static analysis of a program's absolute, or hard, WCET can often be far in excess of the execution times actually realized at run time. Thus, in the context of soft real-time deadlines, it may be worthwhile to alternatively consider a WCET that is not absolute, but at the 99th percentile, or other sufficiently high cutoff value. This value we call the soft WCET. The authors have modified an existing timing analyzer and simulator to determine the execution time distribution between the BCET and WCET. The work in modifying the simulator opened our eyes to the fact that these distributions are quite varied in their width, skewness and maximum value.

However, the major contribution of this paper is the update to the static timing analyzer, which uses the traditional path analysis technique. Rather than computing just a single BCET or WCET value, it can determine execution times at each percentile of probability. The highest value in this distribution is the soft WCET estimate. The approach scales to allow any number of paths. In practice, working with our benchmark suite, the maximum number of paths encountered in the loops was 17. The utility of this work is that the timing analyzer can quickly report both an absolute WCET and a soft WCET. If there is a large difference between these two values, and the application resides in a soft real-time system, then a variety of measures can be undertaken to exploit this gap. For example, dynamic voltage scaling techniques can be used to reduce the clock speed and thus significantly reduce the energy consumption [7]. In a heterogeneous multicore system, the application can be assigned to a slower processor. Finally, if the application is performing some iterative computation, more iterations can be safely performed.

9. Acknowledgements

The authors are grateful for the contributions of two of our colleagues. Colby Watkins implemented the graphical user interface that integrates the output from both the timing analyzer and hardware simulator. William Graeber ported the GUI driver to PHP, making it possible analyze code from a remote site without having to install the timing tools. The authors also thank Peter Puschner and Iain Bate for their helpful suggestions. This research was supported in part by the Furman Advantage Research Program.

10. References

[1] ATLAS, A. K. and BESTAVROS, A. „Statistical Rate Monotonic Scheduling," *Proceedings of the IEEE Real-Time Systems Symposium*, December 1998, pp. 123-132.

[2] BERNAT, G., COLIN, A., PETTERS, S., „WCET Analysis of Probabilistic Hard Real-Time Systems," *Proceedings of the IEEE Real-Time Systems Symposium*, December 2002, pp. 279-288.

[3] COFFMAN, J., HEALY, C., MUELLER, F., WHALLEY, D, „Generalizing Parametric Timing Analysis," *Proceedings of the ACM SIGPLAN Conference on Languages, Compilers and Tools for Embedded Systems*, June 2007, pp. 152-154.

[4] EDGAR, S. F. *Estimation of Worst-Case Execution Time Using Statistical Analysis*, PhD Thesis, University of York, 2002.

[5] HEALY, C., ARNOLD, R., MUELLER, F., WHALLEY, D., HARMON, M., „Bounding Pipeline and Instruction Cache Performance," *IEEE Transactions on Computers*, January 1999, pp. 53-70.

[6] HEALY, C. and WHALLEY, D., „Automatic Detection and Exploitation of Branch Constraints for Timing Analysis," *IEEE Transactions on Software Engineering*, August 2002, pp. 763-781.

[7] MOHAN, S., MUELLER, F., HAWKINS, W., ROOT, M., HEALY, C., WHALLEY, D., „Parascale: Exploiting Parametric Timing Analysis for Real-Time Schedulers and Dynamic Voltage Scaling," *Proceedings of the IEEE Real-Time Systems Symposium*, December 2005, pp. 233-242.

[8] ROSS, S., *A First Course in Probability*, Macmillian, New York, 1988.

[9] WISHART, J. „Cumulants of Multivariate Multinomial Distributions," *Biometrica* **36** (1/2) June 1949, pp. 47-58.