How much Worst Case is Needed in WCET Estimation? *

Stefan M. Petters
Department of Computer Science
University of York
United Kingdom
Stefan.Petters@cs.york.ac.uk

Abstract

Probabilistic methods provide probability density functions for the execution time or the assumed worst case execution time instead of a single WCET value. While the resulting probability tends to fall towards zero quickly, the actual zero value, i.e. the 100 % guarantee, is reached only with unreasonable overestimation of the real WCET. In order to cope with this, this paper proposes to use similar techniques to hardware dependability analysis, where a 100 % guarantee is physically impossible and a certain, usually very small amount of risk is acceptable.

1 Motivation

Modern high performance processors include many features which usually make cycle true simulation infeasible. As a result this either imposes impractical limitations on the code or operating system to allow for WCET estimation, or the methods used to capture these effects have to introduce simplifications that lead to results which may be up to an order of magnitude beyond the physical possible WCET.

One possibility to get around this problem is the deployment of statistical methods. Throughout this paper the validity of these methods is assumed. Additionally it has to be assumed that the methods provide a description of the program behaviour which correctly covers the execution time for all modi of operation. A major problem of such approaches is that they result in approximations of the (worst case) execution time, whose probabilities are non-zero, but very small for a long way beyond the physical WCET. The question now is, whether one has to go for the zero-probability of an error, which tends to be as pessimistic as static WCET analysis, or if a probabilistic guarantee suffices. While the first case has no real advantage compared to static WCET analysis, the second has the open issue as to which probabilistic guarantee to accept as good.

*The work presented in this paper is supported by the *European Union* as part of the research programme "Next TTA"

2 Probabilistic WCET Analysis

Research in probabilistic WCET analysis can be divided in two categories:

- Appraches using observed test cases to reason about the probability of an execution time not observed during the tests.

- Approaches analysing small parts of the program in order to reason about the probability of different combinations of the results of the smaller units.

As there are currently no publications in the second area, we will focus on an example of the first research area. Stewart Edgar uses a black box approach in [1]. The description of the method here can only be very coarse and the reader should have a look at the original papers on this topic (e.g. [1, 2]).

The program is run several times, with random input data and the end-to-end execution time of the program runs are measured. As it is obvious, the measurements will not likely include the physical WCET of the program on that processor in the general case, extreme value statistics are deployed to reason about the execution time longer than any experienced during measurement. Extreme value statistics are concerned with modelling the right and/or left hand tail of a probability distribution, as opposed to the modelling of the average case with conventional statistics. This induces that outliers in the measurement data, which are usually disregarded with conventional statistics, have considerable impact on the modelling parameters of extreme value probability density functions.

Extreme value statistics are well known in the area of financial risk assessment and civil engineering. In the latter case the assessment of maximum wind speeds or flood levels is computed utilising these technique in order to dimension the statics of buildings. There are three type of extreme value probability density functions, described with a theorem which corresponds to the central limit theorem of the normal distributions. As a necessary precondition to apply this technique, the underlying random variables have to be independent and identically distributed.

For the approach the most simple solution of a *Gumble distribution* has been chosen. This model only uses deviation and mean of the random variable, in our case the observed execution time. The following equation show the Gumble probability density function and the cumulative Gumble probability density function:

$$g(t) = exp\left(-\left(e^{-\frac{t-\mu}{\sigma}} + \frac{t-\mu}{\sigma}\right)\right)\frac{\mu}{\sigma} \qquad (1)$$

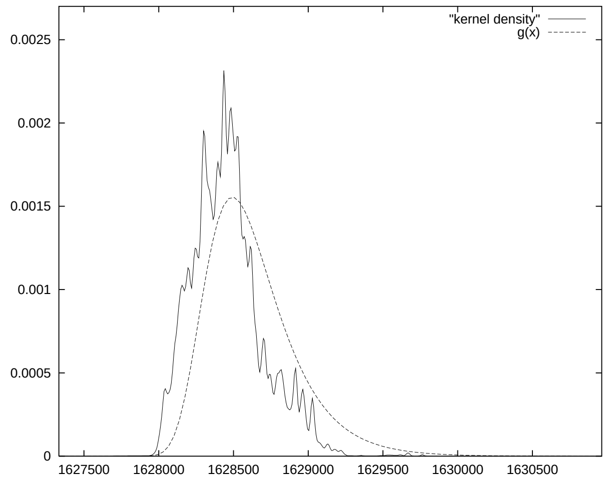

Figure 1: Sample Measurement Data and Extreme Value Approximation.

$$G(t) = exp\left(-e^{-\frac{t-\mu}{\sigma}}\right) \qquad (2)$$

The cumulative variant expresses the probability of an execution time below the value t.

An example execution time measured and the corresponding Gumble distribution is given in figure 1. The measured times are given in a kernel density transformed representation. The transformation is used to display discrete data as a continuous curve and thus allowing the comparison by inspection with the extreme value approximation.

A major drawback of using the Gumble distribution to approximation is the non-zero probability for execution times, except for $\pm\infty$. While the probability of the execution time exceeding 20σ beyond the mean is only $2.06E^{-9}$, the “risk” is still there. As experiments show, this probability reaches quickly $1.0E^{-20}$ and less with an overestimation of some 10% (cf. [3]).

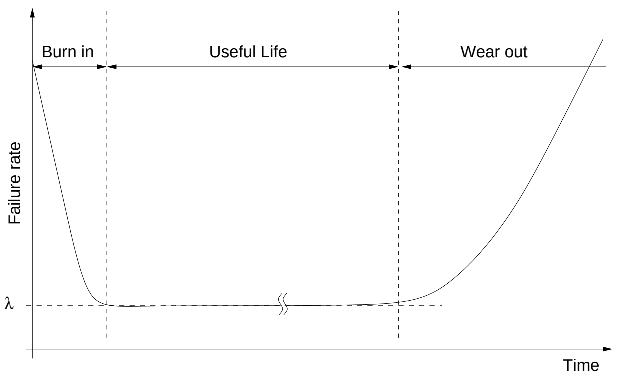

Figure 2: Typical Variation of Fault Ratio of Hardware Components over Time [4].

3 Hardware Considerations

This section will give a short introduction in the mechanisms to risk assessment of hardware components. Figure 2 shows the typical distribution of hardware faults in electronic equipment over time.

During the period of *burn in* the probability of hardware failure is higher, due to faults in the productions of the components. A good example for such a behaviour are errors due to the statistic deviation in the doting of semiconductors. To avoid the high probability of failures in the *burn in* period, the components are in general case run for a time before deployment in a dependable system to weed out bad components. This process is in most cases speeded up by undertaking this testing phase under more extreme circumstances than the system has to endure in real operation (e.g. heat, cold, mechanical stress). Thus a production error that might show up only after months or years down the line is uncovered after a few hours or days of operation.

After the burn in time, the hardware components reach a more or less constant failure rate of λ. Usually this useful lifetime is quite long. In the end the wear out sets in, where, for example, saturation effects[1] in the semiconductor set in. The failure rate λ after burn in as well as the average life time of a given hardware component is usually known. The reliability $R(t)$ of a component not to fail is given in equation 3.

$$R(t) = e^{-\lambda t} \tag{3}$$

For the computation of system failure usually the *mean time to failure* (i.e. λ) is taken to compute the overall systems failure rate. Since the usual failure rate is less then one failure in the lifetime (T_{life}) of a product, the failure rate can be transformed into a failure probability (p_{life}) for the lifetime of the system. This failure probability can be computed using equation 4.

$$p_{\mathsf{life}} = \int_0^{T_{\mathsf{life}}} R(t)dt \tag{4}$$

A similar reasoning may also be applied to software components. The major difference between software and hardware components is the discrete nature of failures of the software components as opposed to the continuous nature of failures of the hardware components. Assuming the program has no algorithmic errors, exceeding a computation time alloted to the program can be considered a software failure in real-time systems. A basic requirement for this is the assumption that the probability for an overrun of the alloted time for an individual run p_{excess} is known and constant for all runs. Additionally the maximum amount of task releases for any given time is essential for the computation. This is usually defined as a minimal inter arrival time T_{task}

The probability of a failure over the lifetime of the product is computed using equation 5.

$$p_{\mathsf{life}} = 1 - (1 - p_{\mathsf{excess}})^{\frac{T_{\mathsf{life}}}{T_{\mathsf{task}}}} \tag{5}$$

Defining an acceptable failure probability during the lifetime, which would be in the order of magnitude of

[1] One problem in the semiconductor industry is that the doting of semiconductors may be done by diffusion and these donated atoms tend to start drifting inside the semiconductor.

the failure probability of an hardware failure, it is easy to compute an acceptable p_{excess} transforming equation 5 into:

$$p_{\text{excess}} = 1 - p_{\text{life}}^{\frac{T_{\text{task}}}{T_{\text{life}}}} \tag{6}$$

4 Conclusion

While the number of publications in the area of probabilistic WCET estimation is quite limited up to now, the number of people working on this issue is becoming larger. Interpreting an overrun of an assumed value for the WCET of a program as a software fault, similar probabilistic techniques as for hardware component failures may be used. This is particular useful whenever probabilistic methods are utilised to reason about the WCET, as these methods tend to provide probability density functions to describe the WCET instead of a single value. The validity and applicability of this method is subject to discussion.

References

[1] A. Burns and S. Edgar, "Statistical analysis of WCET for scheduling," in *Proc. of the IEEE Real–Time Systems Symposium (RTSS'01)*, (London, United Kingdom), Dec. 4–6 2001.

[2] A. Burns and S. Edgar, "Predicting computation time for advanced processor architectures," in *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, (Stockholm, Sweden), June 19–21 2000.

[3] S. M. Petters, *Worst Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute of Real–Time Computer Systems, Technische Universität München, Munich, Germany, 2002.

[4] N. Storey, *Safety-Critical Computer Systems*. Addison–Wesley Publishing Company, 1996.