Predicting Computation Time for Advanced Processor Architectures

Alan Burns and Stewart Edgar*
Real–Time Systems Research Group
Department of Computer Science
University of York
United Kingdom
e-mail: {burns,sfe}@cs.york.ac.uk

Abstract

Estimating computation times using analysis techniques is always safe but is becoming prohibitively complex or pessimistic with modern processors. The only alternative approach is to use measurement, but this has the significant disadvantage of optimism – the largest value seen during testing may not be the largest experienced during deployment. In this paper we subject data obtained from measurement to statistical analysis using the techniques of extreme value estimation. A simple case study is described and the approach is illustrated via this study which focuses on the superscalar technique of branch prediction. The approach is applicable to all forms of hardware-induced temporal variability.

1 Introduction

The Liu and Layland model [7] of periodic tasks in a scheduling problem has as one of its requirements a priori knowledge of computation time, which is to remain fixed. This can be extended easily by replacing this value of fixed computation with a known *worst-case execution time* (WCET), the worst performance of a task over all possible situations in which it executes. Since the purpose of a schedule is to present a provably safe method of guaranteeing the completion of execution, this value has to be pessimistically estimated resulting in the unnecessary addition of slack to a schedule. The accuracy and efficiency of a schedule is dependent on the information used to represent the system and the tasks executing on it. Hence it is preferable to obtain a value for the WCET by careful analysis of hardware and software to ensure its correctness [8].

For software in particular, the value of the WCET will be dependent on the nature of the task and the way in which it is executed on the hardware. In systems involving advanced off-the-shelf components this analysis may be difficult to perform and impossible to guarantee correct due to the complexity of the analysis, or to lack of knowledge of the operation of components. A possible method is to substitute measurement and modeling for analysis. Since measurement cannot find the WCET without exhaustive execution under all possible environments, any estimated WCET found by this method will only have a probability of success. This raises the need for statistical modeling, where computation time is represented by a probability function and WCET estimates have an associated level of confidence with which the estimate can be held. Choosing measurement above analysis also allows us to abstract away from the structure of the processor - the method of execution does not need to be known and the same techniques can be applied to a number of processors.

In section 2 previous work in this area is presented including an introduction to the statistical techniques required for modeling computation time, together with a brief explanation of why they were chosen. Section 3 describes how these techniques were applied to data obtained from an experimental test-bed to estimate a value for the WCET. The results of this test and the subsequently derived knowledge are presented in section 4, together with a review of the statistical techniques. The conclusions that can be drawn are presented in section 5.

*This work has been partially funded by the EPSRC.

2 Statistical Techniques

There are three major steps when using statistics in any field. Firstly data has to be obtained. Secondly these data have to be collated, either using mathematical formulae or by finding a suitable model. Thirdly this model is used to generalize the behaviour of a system beyond the existing data. The method of obtaining data must be carefully designed to prevent the variation in data observed in the sample from being caused by effects outside the domain of interest for the study.

The major difficulty in statistics is the construction of a suitable model from known data. Typically the chosen method of modeling any data is to use a particular *distribution*, a mathematical function assigning probability values to a set of events, usually with a continuous order. For each distribution function there will be a corresponding *density function* representing the density of data samples against the magnitude of the data. Random data samples, such as height, tend to form characteristic bell shaped density curves indicating a central clustering of samples around a mean with decreasing density away from this typical value, as can be seen in figure 1. This distribution does not occur randomly, rather it is a consequence of data modeling. The *Gaussian (or normal) distribution* for mean μ, variance σ^2 is denoted $\mathbf{N}(\mu, \sigma^2)$ and has density function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right)$$

This distribution not only has a similar bell-shaped curve, but it is also provably correct for modeling random data samples, as proved by the following standard theorem:

Theorem 1 (Central Limit Theorem) *If a random sample of n data items, $X_1 \ldots X_n$, are independent and identically distributed (iid) with a mean μ and a variance σ^2 then*

$$\frac{(X_1 + \cdots + X_n - n\mu)}{\sqrt{n\sigma^2}} \to \mathbf{N}(0,1)$$

That is to say, for a sample of data taken from any source which can be shown to be produced under unchanging conditions can be approximated by the standard normal distribution (whose $\mu = 0$ and $\sigma^2 = 1$). The larger the sample the more accurate the approximation becomes. The values of μ and σ^2 can be easily calculated from the sample of data.

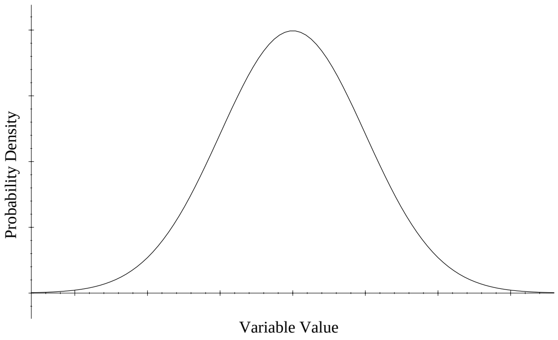

Figure 1: An example of the bell-shaped Gaussian density function.

These techniques are useful when the data we are interested in modeling are themselves relatively close to the mean. By the nature of traditional statistics any outlying samples are deemed to be less significant become they do not have as great a density of numbers. Conversely when modeling computation time for the purposes of scheduling the more extreme samples are very important as they (in the maximal case) correspond to the instances of a task which are likely to have bearing on the calculation of a WCET. Hence it is important to know the quality of the data that is to be modeled. For the purposes of estimating a WCET, this quality is not a behaviour of the sample as a whole, but the amount of computation that should be assigned to a task so that it will execute when the demand for computational resources is high. Modeling the maximum of a data sample requires a different approach to statistics.

As with the normal distribution, the distributions that accurately model the maximum of a distribution are not chosen randomly, but are selected so they possess an important property which allows modeling for all distributions generally. These distributions are known as *extreme value* (EV) distributions and contain the following definitions:

Gumbel	$G_0(x) := \exp(-e^{-x})$	
Fréchet	$G_{1,\alpha}(x) := \exp(-x^{-\alpha})$,	$\alpha > 0$
Weibull	$G_{2,\alpha}(x) := \exp(-(-x)^{-\alpha})$,	$\alpha < 0$

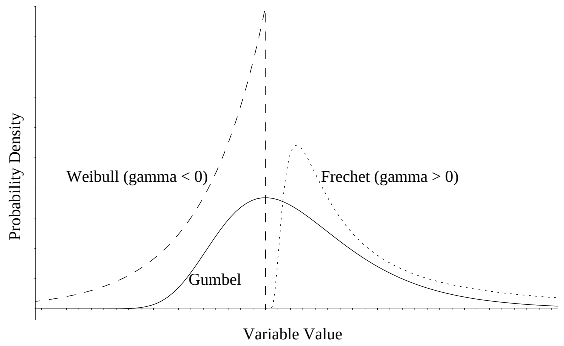

Figure 2: Plot of EV density functions with standard parameters.

Although there are three different distributions with the correct properties for the purposes of modeling, they can be expressed as a single equation: $G_\gamma(x) := \exp(-(1+\gamma x)^{-1/\gamma})$ with the restriction that $1 + \gamma x > 0$. This property is expressed by an analogy of the Central Limit Theorem:

Theorem 2 (Fisher–Tippett) *If $F(ax+b)$ has a (non-degenerate) limiting distribution function for constants $a > 0$ and b as $n \to \infty$ then:*

$$|F^n(x) - G((x - \mu_n)/\sigma_n))| \to 0 \text{ as } n \to \infty$$

for an EV function G and parameters μ_n and $\sigma_n > 0$ drawn from the sample distribution [3].

When using these distributions it is important to remember that we are modeling a specific feature of the data, the maximum. For example; the probability that an extreme value statistic X is greater than a time t is denoted by $P(X > t)$. If the system modeled by X has to have a 99% likelihood of executing within this time–limit t then this requirement is valid if the value of t satisfies the equation $P(X > t) \leq 0.01$.

As the model for a sample of data is a clearly defined equation, the exact model of the maximum computation time is described solely by the parameters. These are the mean μ and variance σ^2, calculated from the sample, and a shape parameter, either α or γ, depending on the model chosen. This later parameter can be calculated using a *maximum likelihood estimator*, a function dependent on the model chosen, which will produce a value for a parameter which produces the most likely model for a sample of data. The likelihood function for any probability density function f and parameter θ is $L(x, \theta) = f(x; \theta)$, whose maximum is the solution of the equation $dL/d\theta$ [6]. The choice of distribution and parameters together give a probabilistic model of a system which allows us to inquire about a task in a schedulability problem when it is performing at, or beyond, the limit of the available resources. These techniques will be shown with a simple case study allowing an initial example of their possible application.

3 Case Study

To justify a statistical model, it is first necessary to explain why and how variability in computation arises. Processors are no longer simple mechanistic devices that will execute instructions singularly and without variation. To prevent the speed of memory affecting the speed of the processor core, a cache is introduced to buffer instructions and data. Execution will vary as to whether these data are in the cache or whether they have to be retrieved from main memory. In order to increase the throughput of instructions, the individual commands can be split into stages and different stages of successive instructions executed simultaneously. This process is known as a pipeline, but causes problems when branches have to be taken within programs, as the following instruction to be executed can depend on the previous result. The pipeline is stalled resulting in a drop in performance. Some form of *branch prediction* can be employed to regain some of this lost time, but it results in behaviour that can be difficult to predict. These *superscalar* architecture features are employed on most modern processors. Although they result in an improvement in overall performance, they also make the exact timing of programs executed on them non-deterministic and possibly unpredictable.

To measure the effect of superscalar features on the response time of programs that are executed through such a processor, we need a test-bed containing a processor that exhibits these features. This test-bed must be experimentally designed to isolate the variability caused by architecture design away from the unpredictability caused by the rest of system or by the measurement process itself. Secondly we require a suitable timing system that will measure the time taken for a program with a predefined start and stop to execute to within a suitable degree of accuracy. Thirdly we require a program that will make use of the superscalar features of the processor, but will also reflect the tasks that a real world system would have to face. Together these three features will produce an experimental test-bed that allows us to reliably obtain timing data from a task present in a real–time system.

3.1 High–Level Analysis

It is the variability induced by superscalar architecture features that we are interested in modeling, rather than variability caused by the flow of control through application software. This paper assumes that such problems can be address by standard high-level analysis techniques [2, 8]. An assumption of this paper is that the knowledge required to perform this analysis is limited and that the gaps must be filled if a value for the WCET is to be obtained. The techniques for constructing this analysis are not made redundant by statistical modeling, but instead enable the modeling to be applied to the smallest possible sections of code. This may be a basic block of control flow, which is independent of data and the path through a programme, or a block which has only one or neither of these qualities [9]. Since it is an assumption that the flow of the programme or the influence of the data would be factored into the analysis if known, the only method of handling these difficulties is to use statistical modeling. For the purposes of this paper the test code is a block of instructions, and that the analysis of control flow has allowed us to consider this segment of code in isolation. This allows the presentation of the statistical theory with a sufficiently manageable example, but also, when combined with similar analysis of other superscalar features allow a comprehensive consideration of variability in a system due to hardware factors.

3.2 Experimental Test-bed

The first requirement, scientific isolation, is fulfilled by producing a test-bed consisting of two distinct components. The first component is a machine that contains the processor to be measured, in this case an Intel Pentium II processor. This computer also contains a timer board containing a 48-bit real–time clock with interrupt or software control. With software controlled operation it is able to measure the time between activation and a signal stopping the timer to within an accuracy of about 20-30 clock ticks, for this processor this equates to about 2 microseconds. Any attempt to write a result to either the screen, memory or disk would impact on the time measured and hence the conditions under which the experiment took place. To avoid this problem a second machine is employed to record the result, sent down a serial connection. Apart from the simple task of pushing information down the serial connection, which is of constant cost, the sole work performed by the first computer is for the result of the experiment to be produced. Hence the conditions of the experiment are suitable for the scientific investigation of a suitable timing problem.

The subject of the experiment was a Pentium II processor running at 233MHz with 32K of first level (L1) cache and 512K of second level (L2) cache. Despite the size of the cache, it is possible for a task to involve cache misses during execution. For guaranteed performance any large scale task would require the assumption of possible cache misses, when these events are likely to be relatively infrequent. The superscalar features of the processor that need to be tested include a variety of performance enhancing techniques. The Pentium II has hardware support for out-of-order execution. If a micro-operation requires additional time from the standard assigned to it, for example an access to main memory, the processor will allow following instructions to jump its place in the queue if this optimization will not affect the integrity of the data. This allows a task to execute in a smaller interval of time, yet makes the job of analysis to ascertain this value all the more complex as we require knowledge not only of the task and the method in which it executes, but also the platform on which the execution is to occur either of which may be unknown to the observer [11, 10].

In the superscalar architecture of the Pentium II instructions are pipelined, and so there are difficulties with branch instructions. Since there are two possible paths that can be taken it is necessary to make this decision before the next instruction can affect the permanent state of the processor. The simplest method is to stall the pipeline, i.e. prevent any instruction from passing through the pipeline before the branch decision is made. This will result in a performance disadvantage. The solution is to perform *branch prediction*, where the processor, at run time, makes a prediction as to which branch to take. In the Pentium II this prediction is based on the past behaviour of the program at this particular branch, stored in the Branch Target Buffer (BTB) for the last 512 branches encountered, in a section of on-chip hardware. If an instruction begins to be executed but has been incorrectly predicted then the instruction can be aborted and the pipeline flushed as no permanent change has been made. The worst-case performance is no worse than if the pipeline had been stalled, thus there is a resulting net improvement in the time taken for micro-instructions to be processed. However this speed up is not guaranteed. While there will be an upper limit on the number of failed branches the analysis can become infeasible without assuming that a lower limit on the number of branches that can be mistakenly predicted. Hence there will be a difference between the worst case execution, where all branches produce this worst case behaviour, and the expected case where a branch instruction appears as an implementation of a loop or a repeated `if...then...else` structure where a branch is likely to take the same path on repeated occasions.

3.3 Experimental Structure

To measure the effect of branch prediction on computation time we need a program whose computation is known to be fixed, apart from the additional cost resulting from incorrect branch prediction. This program should also be designed to prevent the contamination of results by other superscalar features. By structuring the programme in such a way this contribution to variability in the system can be eliminated. There is no problem with caching since the size of both the microcode and associated array are of a vastly smaller degree than the size of the cache. Prior execution to reset the BTB ensures that both code and array will be in L1 cache before timed experimentation takes place.

The method used to achieve both of these requirements is to have an iterative program whose structure is simple enough to be analyzed at the microcode level and small enough to fit in cache. The actual work that the program does is unimportant as long as it can be guaranteed to be consistent. The program chosen as the subject was a bubble-sort algorithm. Although this satisfies the requirements of a large number of branch instructions (of order N^2 for an array of size N), the number of incorrect branch predictions is not itself random. As the array becomes more ordered the number of incorrectly taken branches falls. Consequently the program was changed to a single pass through an array whose elements are populated by compiled instances of the C `rand()` procedure call. A pseudo-code summary of the experiment itself is presented in figure 3.

For correct experimental procedure it must be known that the requirements detailed above are suitably addressed. The program is checked at the microcode level for expected behaviour. The instruction cost for either branch is identical consisting of four instructions to move data between memory and registers. Either branch is of the form:

```
mov ecx,ebx          Put B=a[i] into C
mov ebx,+4H[eax]     Put a[i+1] into B
mov [eax],ebx        Put B into a[i]
mov +4H[eax],ecx     Put C into a[i+1]
```

where B (register `ebx`) is the holding register and C (register `ecx`) is the temporary register. All four `mov` instructions act in a temporally fixed manner – the pipeline improves performance, but for these instructions does not affect variability.

The operating system that the program runs on is QNX [4], a real–time UNIX–like operating system which allows test code to run at a higher priority than the operating system or any other tasks, preventing outside influence in the performance of the test code.

```
Assign task highest priority.
Save current interrupt status.
Reset timer.

for ( X number of experiments )
    {
    Assign integers to an array
    Reset and start real-time clock

    for ( i < Size of array )
        if ( a[i+1] < a[i] )
            { Swap a[i] and a[i+1] }
        else
            { Swap a[i] and a[i+1] }

    Stop and read real-time clock
    Write clock value to serial port
    }
```

Figure 3: Pseudo-code for the test program.

The size of the array is arbitrary, but should be chosen large enough so that the results obtained are above the granularity of the timer.

4 Experimental Results

The results of the experiment were a series of samples of timing data with the largest data sample being 4527 clock ticks and the least 4476 clock ticks, a variation of 51 clock ticks (approximately 30 milliseconds) and above the accuracy of the timer board. The results are displayed as a *kernel density* a method of transforming discrete data into a continuous curve which can be visually compared to the approximating curve of any particular model. Figure 4 allows a visual comparison between the kernel density and an attempt to model this data using a Gaussian distribution.

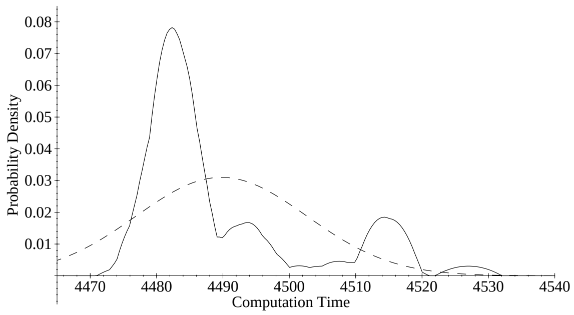

Figure 4: Density functions of data sample and the inaccurate Gaussian approximation.

The kernel density of the sample is represented as a solid line while the Gaussian approximation, the model chosen in standard statistics for the purposes of modeling is represented by the dashed curve and has parameters of mean, $\mu = 4483$ and variance $\sigma = 5.8$. The two curves represent the density of data sample for variations in computation time. It is important to remember that the kernel density is the data itself and the Gaussian approximation is the model. Visually it can be seen from the graph that the approximation is not accurate, with coinciding peaks, but poor correlation for spread. Even so this graph and model can still convey important information about the data from the test-bed. The difference between the greatest sample data item (the greatest observed computation time of 4527 clock ticks) and the statistical average is not relatively great when compared to the overall computation time. Conversely, using a 90%-quantile (that value for which the Gaussian approximation shows a 90% probability of being less than) gives a value of about 4500 clock ticks. This shows that construction of any detailed information based on a standard statistical technique, *i.e.* a model constructed in accordance with the Central Limit Theorem of section 2, will be unable to fully respond to the needs of a problem that requires accurate modeling at uncommon events that have values deviating from the mean to a large degree. The Gaussian method of approximation is not only mathematically inferior to the method of extreme statistics, it has been shown to be so for this example.

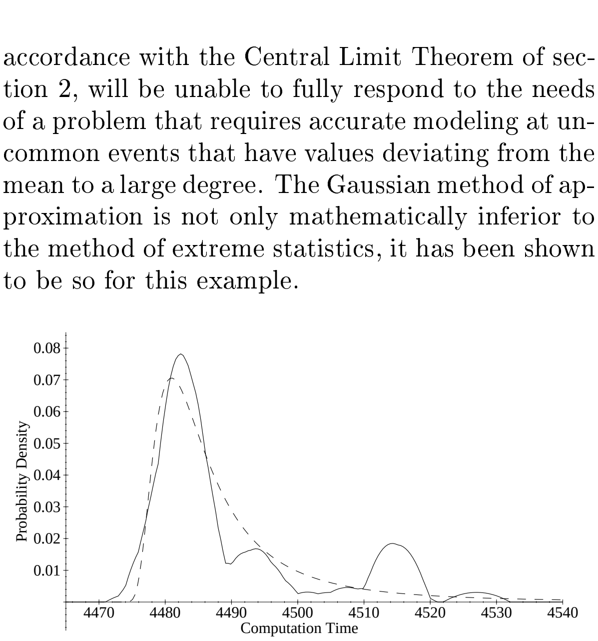

Figure 5: Density functions of data sample and EV approximation.

Instead of modeling the computation time and then using this model to find features of the system that we are interested in, it is preferable to choose those particular features and model them directly from the dataset. This is where the techniques of extreme statistics presented in section 2 can be applied to provide an improved method of approaching the problem. In particular the WCET can be estimated using one of the equations shown to be applicable to this field of statistics by application of the Fisher–Tippett theorem. The model for the maximum of a task based on a sample of data is optimally chosen from the family of extreme value distributions as defined in section 2 [3]. The graph in figure 5 represents the extreme value density function whose parameters have been chosen by the maximum likelihood method. Theory dictates this to be a better approximation, and the sample data from which this approximation has been developed supports this view. The density function shown in figure 5 is a model of the maximal value that would be observed in a hypothetical data sample. Solving the equation $P(\text{Observed CT} < X) = \int_0^\infty G.dt = n$ for the value of X will give a confidence level of $(n/100)\%$ for the value of X being the greatest value that would be observed in the execution of the system. These values have been collated for varying computation time in figure 6. Note that this diagram also contains a dotted line representing the data sample, which provides an additional visual check on the accuracy of the model.

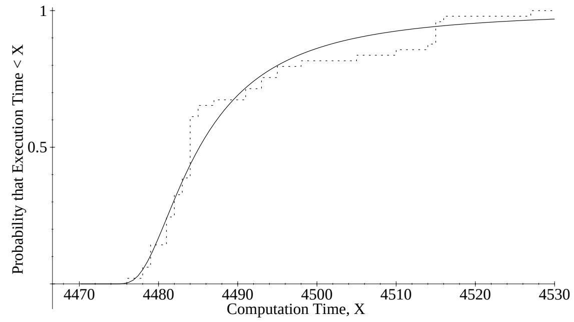

Figure 6: Graph of confidence in estimate against computation time, compared to sample distribution function.

4.1 Inferences

The graph for figure 6 and the underlying mathematics allows the calculation of the probability with which an estimation of computation time t will be adequate at run–time. This is not a probabilistic assessment of the WCET, however, merely a probabilistic assessment of the likelihood that a computation time will exceed t. However the same model can be used to discover the probability of computation time exceeding the greatest observed data sample. In the case study this value is 0.965 (to 3 significant figures), showing that there is already a 96.5% probability that computation time will not exceed the maximum already encountered. If this is to be extended to cover an estimation of computation time beyond the values already encountered this figure can then be used to find the *conditional probability* of the computation time. This is illustrated in figure 7.

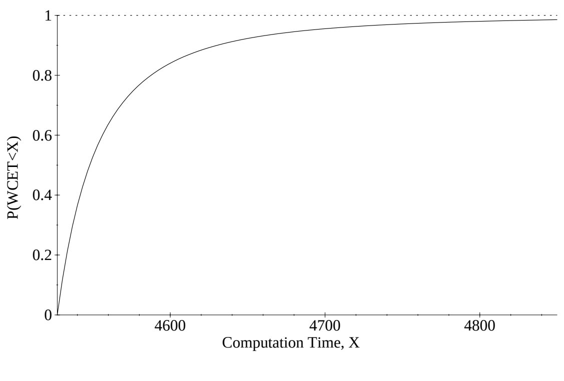

Figure 7: Graph of conditional probability.

Success Rate	Probability of Overrun	Estimated WCET
0.9	10^{-1}	4505
0.99	10^{-2}	4570
0.999	10^{-3}	4749
0.9999	10^{-4}	5250
0.99999	10^{-5}	6651
0.999999	10^{-6}	10570

Table 1: Estimates for WCET with associated probabilistic guarantees.

From this diagram it is possible to obtain values for the probability of various estimations of the WCET being correct. These are displayed in table 4.1 for increasing orders of magnitude. The graph of conditional probability is logarithmic, as can be seen in figure 7, so it is conversely true that the time penalty for increased magnitudes of likely success is exponential. Although the value of 4670 as guarantee for the WCET seems unnecessarily pessimistic, it must be remembered that in general the conditions under which a task will execute are unknown, beyond those specified by the environment, and it is inappropriate to argue that such a time is unnecessarily large.

5 Conclusions

This paper has described the various features of modern processor architectures that may prevent direct analysis from being a viable option. An alternative method has been presented involving statistical modeling. It was shown to be valid statistically from established theory and a presented example shows that this theory is applicable to the modeling of computation time. Finally this model was used to show how an estimate of the WCET for a task could be constructed without requiring knowledge of the underlying processes. An advantage of this process is that it is independent of the cause of variability and the operation of the task itself. From a probabilistic estimation of WCET it is possible to provide scheduling analysis that is again probabilistic in nature. This is however beyond the scope of this paper.

References

[1] A. A. Balkema and L. de Haan. Residual life time at great age. *Ann. Probab.* 2, pp. 792–804, 1974.

[2] R. Chapman, A. Burns and A. Wellings. Integrated program proof and worst–case timing analysis of SPARK Ada. *ACM Workshop on language, complier and tool support for real–time systems.* ACM Press, 1994.

[3] R. A. Fisher and L. H. C. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Proceedings of the Cambridge Philosophical Society.* Vol. 24, pp. 180–190, 1928.

[4] D. Hildebrand. An Architectural Overview of QNX. http://www.qnx.com/literature/ whitepapers/archoverview.html

[5] Intel Corporation. Intel Architecture Optimization Reference Manual. Order number 245127-001.

[6] B. F. Kimbell. The bias in certain estimates of the parameters of the extreme-value distribution. *Ann. Math. Stats.*, vol. 27, pp. 758.

[7] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *JACM*, 20(1), pp. 46–61, 1973.

[8] C. Y. Park. Predicting program execution times by analyzing static and dynamic program paths. *Real–Time Systems*, vol. 5, pp. 31–62, 1993.

[9] P. Puschner and A. Schedl. Computing Maximum Task Execution Times – A Graph-Based Approach. *Real-Time Systems*, 13(1):67–91, July 1997.

[10] J. A. Stankovic. Misconceptions About Real-Time Computing. *IEEE Computer*, October 1998, pp. 10–19.

[11] J. Sun and J. Liu. Bounding Completion Times of Jobs with Arbitrary Release Times and Variable Execution Times. *Proc. IEEE Real–Time Systems Symposium 1996*, pp. 2–12.

[12] N. Zhang, A. Burns, and M. Nicholson. Pipelined Processors and Worst-Case Execution Times. *Real-Time Systems*, 5(4):319–343, Oct. 1993.