Video Subset Selection for Measurement Based Worst Case Execution Time Analysis

Sitsofe Wheeler, Iain Bate and Mark Bartlett
Department of Computer Science, University of York
{*sitsofe, iain.bate, bartlett*}*@cs.york.ac.uk*

Abstract

Worst Case Execution Time (WCET) has traditionally approached problems with small, well defined input spaces. For processes with a large input space (such as video) existing techniques struggle to produce a meaningful result. This work investigates a technique that reduces the input space while still preserving execution time properties to allow subsequent WCET analysis to be more effective.

1. Introduction

The Worst Case Execution Time (WCET) of a program can be a key component of modern embedded systems as it is needed for any scheduling decision or analysis that is to be performed [4]. The criticality of correctly identifying the WCET (rather than just one of the longer execution times) depends on the consequences of a longer execution time. A thermal control regulating the temperature in a hotel guest's room may take one second to calculate whether the fan should be turned on or off. However if execution occasionally takes two seconds, such an overrun will probably go unnoticed. In a different scenario such as the thermal control for a nuclear power station, a response may be required within a tenth of a second. In such a situation, taking a fifth of a second to return an answer could be fatal. Some systems (such as a system transmitting a mobile phone call) fall into a middle ground, where missing a deadline is not critical but repeatedly missing a deadline will be noticed and interferes with the device's usage. However, even for such a system there may be the need to reliably predict the likelihood and pattern of missed deadlines. This suggests that where possible, the WCET figure obtained via analysis needs to be a safe value, i.e. larger than the actual WCET. Further, too large a value also causes problems, as resource availability needs to be planned based on the WCET, hence any overestimation wastes resources and consequently money.

Finding the input that generates a program's WCET through algorithmic static analysis is difficult because it is partially a property of the program itself. As such, for an arbitrary program, algorithmically finding its WCET can be seen as a more complicated version of the halting problem (which itself is undecidable) [14]. Attempting to perform exhaustive testing can also be problematic when a program has a large input space. Due to the amount of time it takes for the program to complete, it may be infeasible to run the program with every possible input. However, failing to test every input leaves the possibility that the WCET would be found with one of the untested inputs.

There are two main techniques for determining the WCET of software, static and dynamic analysis, with some research proposing a hybrid of the two [13]. Dynamic analysis involves executing the software by applying test vectors to it, and then analysing the behavioural aspects of the software. There are two main approaches used for dynamic analysis; search-based techniques that use a heuristic to drive the manipulation of the input space such that the WCET is maximised [18] and probabilistic techniques that form a distribution upon which extreme value analysis can be performed [21]. In contrast, static analysis does not involve executing the software. A common issue for both of these approaches is to determine an understanding of what constitutes a representative input space for the software [21].

A more complex example is that of a video codec. Here, the input space consists of two principal parts; the video itself and the configuration parameters provided to the codec (e.g. the target bitrate). In this paper we argue that most WCET analysis problems can be separated into determining a representative input space and then modelling how the WCET is parameterised with respect to its configuration. This type of approach is analogous to a system receiving a stream of information from other sources but the system itself having configuration or control applied to it. Other examples include Electronic Engine Control (EEC) systems, where the

EEC receives a stream of information from the data bus and separately has a number of (internal) configuration settings which can be altered.

In essence, this means that one part of the input space is fixed such that the next stage of analysis can then manipulate the rest of it. This divide and conquer style approach is beneficial for all forms of analysis. For example with probabilistic analysis, reducing the dimensions of the model to be created means that with given resources a more accurate model can be created [8,17]. The main contribution is to provide a means of determining a representative input signal that can be used by any form of WCET analysis. Here a representative signal is considered to have two principal features. Firstly, it contains a set of input values that includes those values leading to the observed WCET. Secondly, its distribution is a sufficiently close match to that observed over a much longer period, i.e. a more comprehensive signal. When performing analysis in a purely dynamic fashion, the longest execution time might be likely to occur but it not guaranteed. By using distribution comparisons a probabilistic assessment can be performed.

In the case of a video codec, the intention would be to identify a short clip (in the order of seconds or minutes) that contains the observed WCET and a similar distribution to a much longer film. However there is an important threat to validity; for the divide and conquer approach to work, it is important that the signal is representative across the whole configuration parameter range that will be searched in the second phase of the work. To the best of our knowledge no other work has addressed this important problem which answers the following research questions:

1. What constitutes a representative signal?
2. How can the representative signal be found?
3. Is the representative signal valid across the range of configuration parameters?

The work in this paper uses the VP8 video codec[1] which is capable of comparing components of a frame against three different reference frames with the aim of reducing the size of video while maintaining visual quality.

Video codecs are used in almost all situations where a sequence of pictures must be stored and retrieved in a digital format. Many of these are classified as embedded systems and can range from recording surveillance of a building to the transmission of TV to set top boxes. Codecs have varying complexity with some targeting high throughput and quality whereas others are trying to achieve the smallest video size while maintaining a reasonable quality.

VP8 is designed to target a range of environments including embedded systems with low latency, real-time video requirements and contains features such as the ability to select quicker encoding methods based on the time remaining until an output frame's deadline. The VP8 codec provides a good test environment as the available input space for video is huge in terms of size and complexity. For the analysis approach to be scalable, a small sample is needed which gives an identifiable worst-case situation to the required statistical confidence, but also gives a representative distribution if dynamic analysis techniques, including probabilistic-based, are to be used.

VP8 is seeing usage on web sites such as YouTube as a part of Google's WebM video format which has led to major web browsers (Firefox, Opera and Chrome) supporting VP8. Further, Google's Android operating system which targets embedded devices such as phones and tablets has support for VP8. Current estimates put Android's market share of new smart phones sales at 33% [5] showing large numbers of devices with VP8 support.

The structure of the paper is as follows. Section 2 provides a literature survey of relevant work. Then in section 3 the problem is described in more detail and our proposed solution established. An evaluation is performed in section 4 and finally the conclusions are given in section 5.

2. Related Work

There are a wide variety of benchmarks that have been proposed for embedded systems research. Three of the more famous ones are MiBench [10], EEMBC[2], and one set specifically intended for WCET analysis [9]. The first two of these both include video codecs thus showing their importance in modern day embedded systems. However, the maintainers of the WCET benchmarks recognise their simplicity in terms of structure, size and input space are not that representative of the needs of the embedded systems market [9].

To date, video codecs have not been considered widely in WCET literature – for the static analysis approaches they are too complex, featuring difficult control flow and high amounts of data dependent execution. Similarly video codecs pose issues for dynamic analysis because their input space is too large. One option is not to consider the input space but instead check that the input stimulates enough of the software as proposed

[1] http://www.webmproject.org/media/pdf/vp8_bitstream.pdf

[2] http://www.eembc.org/

by Betts [1]. This approach considers more than traditional structural coverage metrics and includes hardware metrics such as cache states. There are a number of issues with this, the first being the observability of cache states on real hardware is often difficult to achieve in an unobtrusive way. Secondly, it can be difficult to attain a high percentage of coverage when using traditional coverage metrics, e.g. path and Modified Condition/Decision Coverage (MC/DC) [6] and therefore a more low-level coverage metric is impractical. Finally Khan [11] showed that focussing on specific metrics, e.g. cache coverage, is only effective for certain types of software. For others, specific measures can actually hinder the ability to find the WCET.

Alternative options are the search-based approaches proposed by Tracey [18], Wegener [20] and Khan [11]. All of these approaches showed that search-based approaches could handle complex software irrespective of their control flow and each showed in their own way the importance of having the right search operators [18, 20] or fitness function [11]. However, these approaches do not attack a problem with an input space as large as video. The final category of approaches is static analysis but these often require constraints on the program structure so as to enable all combinations of input to be accounted for when deriving worst case behaviour.

In the simple example of encoding VGA resolution video, traditional search-based approaches would have to evolve solutions over the whole of the input space. In the case of VGA resolutions the content will be 640 by 480 pixels in size. If the video uses 24 bit colour this results in over 5000 billion combinations for a single frame. The VP8 codec has the concept of "golden frames" that are the amalgamation of arbitrary previous frames while the H264 video codec is capable of referring to any of the previous 16 frames during the encoding process, further increasing the number of combinations. The problem quickly becomes insurmountable without a way of providing a smaller representative input stream that includes worst-case scenarios.

Other dynamic analysis approaches include probabilistic WCET (pWCET) analysis of the form described by Colin and Petters [7]. Based on the typical approaches to pWCET, an important factor is ensuring that the distribution obtained is a good fit to the actual execution times of the software under test. A key limitation of these works is the distribution formed is drawn from a sparse portion of the overall problem space, e.g. a whole set of paths could fail to be included, rendering any result meaningless even if statistical confidence and reliability measures are presented. To date, no approach has been proposed for determining what test vectors are needed to ensure the distribution is formed from suitable data. Theoretically this would need to consider all possible video signal combinations. The issue of choosing a statistically similar distribution which contains the frames with the longest observed execution times is the subject of this paper.

3. Problem Statement and Solution

The aim of this paper is to devise an efficient method for finding a contiguous subset of a much longer input stream that contains:

- The observed WCET of the longer stream.
- A sufficiently close distribution to the longer stream.

3.1. Establishing the Framework

A single video file of the five minute cartoon Big Buck Bunny[3] was used as the source for initial tests. The 14315 raw frames of this video are available in a format encoded with a lossless codec thus the video input space which these experiments tested was not influenced by artificial artefacts introduced by the encoding of the source material.

During experiments, the video was encoded with keyframes positioned at fixed 12 frame intervals and every frame's encoding time was recorded. The VP8 codec's ability to reduce encoding time based on CPU usage was disabled to make runs repeatable. To perform the per frame timing, the encoder's source code was instrumented to record the number of processor cycles before and after a frame was encoded. All normal output (such as creating a file to store the encoded video) was disabled to further reduce sources of interference. While such instrumentation involves a degree of interruption compared to hardware probes, such a technique is simpler to perform and less expensive to implement.

The software instrumentation makes use of the PAPI library [3] by resetting the hardware performance counters before frame encoding starts and reading the hardware cycle count into a preallocated array after frame encoding finishes. The overhead of reading and resetting the counters was measured to be 53000 cycles in the worst case and 17000 cycles in the average case. Across all measurements, the minimum number of cycles used to encode a frame was 4931196 thus at worst 1% of the time is spent on the instrumentation and less than 0.5% of the time is spent on instrumentation in the average case.

[3] http://www.bigbuckbunny.org/

A real system was chosen over the use of a simulator due to execution time length – while simulation provides a more transparent and deterministic environment it also imposes a performance cost. The M5 simulator [2] running on the same machine as above was found to encode a single frame 100–1000 times slower than the native machine, which was prohibitive given the number of frames which needed to be encoded. To further reduce encoding time, each frame was resized to 160 by 90 pixels (one quarter of VGA).

The encoding process was run on a 2.4GHz Intel Core 2 desktop running an x86_64 version Linux. The test setup was chosen to try and minimize timing variability:

- The encoding was run with minimal additional processes reducing the probability that a context switch would occur.
- The ability to page memory to disk was disabled as the act of writing inactive memory to disk can introduce large stalls whose length is hard to determine.
- Only one CPU was enabled to eliminate any impact on execution times due to the process being migrated to a different CPU part way through a run. Additionally this change ensures that all tests were run on the same CPU.
- The CPU speed was fixed to the highest setting. A fluctuating CPU speed can impact the number of cycles a program takes to finish because memory access times remain the same thus creating a different execution time schedule (for example transitioning from a low speed to a high speed shortly after the program is started for the first time). Fixing the CPU speed to one setting ensured this could not occur.
- The process priority was set as high as possible to reduce the number of other processes that could cause preemption.
- The system was rebooted between each change of parameters. This resets the pipeline and cache states creating a similar starting environment across multiple runs.
- Randomisation of the address space location that the binary was loaded at was disabled. This helps reduce the variability due to differing cache access patterns.

To record the time used to run the process, hardware cycle counts are queried before and after each frame is processed. The hardware cycle count is available in per process form, thus offering a resolution finer than a nanosecond when the CPU clock speed is 1GHz or higher. This provides an improvement in granularity when compared to looking at the process' current CPU usage using $getrusage()$ (which is only capable of returning times to the nearest millisecond). Overestimation of execution time is also reduced compared to using the realtime clock because the time the program was not running (if a context switch should take place) is not included in the cycle count.

Tsafrir et al. [19] demonstrate that a large amount of variability in execution times is caused due to interrupts. The majority of this is attributed to periodic timer ticks, although interrupts due to the network card were also recorded. To mitigate this, the kernel's periodic timer was set to its minimum value of 100Hz thus reducing the timer ticks. Further, most device drivers (such as the drivers for the network and graphics cards) were disabled to eliminate as many external sources of interrupts as possible.

The following list describes the encoding parameters that were varied between different encoding runs.

- The enabling/disabling of error resilience (ER).
- The target bitrate (BR). Varies from 1–65536.
- The minimum quantization (MINQ). Varies from 0–63.
- The maximum quantization (MAXQ). Varies from minimum quantization to 63.
- Whether the choice to reduce frame sizes through scaling is allowed (FRS).

The parameter sets tested are described in tables 1 and 2.

3.2. Proposed Method

The proposed method has four phases:

1. The whole video is encoded with each parameter set.
2. A frequency distribution of frame encoding times is composed by collecting execution times into 20 equally sized bins.
3. For a given number of frames an algorithm is applied to find the subset of frames that is most similar to the original while still being contiguous.
4. Stage 3 is repeated for increasingly smaller window sizes.

The algorithm (outlined in algorithm 1) searches for the best contiguous frame subset in the video but using differing numbers of frames. An initial "window" containing 85% of the total frames (as this is 85% of a "100% window") was established and the binned distribution of frame cycle times within it tested for similarity against the full video distribution.

Algorithm 1 Search for good subsets

Input: $frames$
Output: $best[]$

```
1:  WC_p ← pos_longest(frames)
2:  win ← frames
3:  while win.size ≥ 100 do
4:      win.size ← win.size * 0.85
5:      first ← max(0, WC_p − win.size + 1)
6:      last ← min(WC_p, frames.size − win.size + 1)
7:      best[win.size].emd ← 1
8:      for pos = first to last do
9:          win.start ← pos
10:         score ← EMD (win, frames)
11:         if score < best[win.size].emd then
12:             best[win.size] ← win
13:             best[win.size].emd ← score
14:         end if
15:     end for
16: end while
```

The similarity test is performed using the Earth Mover's Distance (EMD) which expresses the amount of energy required to convert one distribution into another when the distributions being compared have the same integral [12]. Each distribution can be viewed as a mound of earth with the ultimate aim of making a column in the first distribution the same height as the equivalent column in the second distribution. To achieve this, earth is taken/deposited from other columns in the first distribution. Those columns in turn will need to be made the same height as their equivalent column in the second distribution. Thus the EMD calculation can be performed by solving an instance of the transportation problem.

As the measurements are recorded in cycle times they are precise, necessitating the use of binning to reduce sensitivity to noise in the data collection process. Additionally, the EMD metric is shown to yield good results when using a small number of bins [15] helping to reduce calculation time.

In this experiment the EMD calculation is simplified by only summing the difference between the equivalent bins in both distributions. If no energy is required to transform one distribution into another then the distributions must be identical as this is the only scenario where no work would need to be performed. Conversely, if a high degree of energy is required to convert one distribution to another the distributions are dissimilar.

To achieve the integral constraint the distributions are normalised by the number of values before the comparison is performed. Multiple starting offsets are tested for each of the subset window sizes, with the offset that is most similar being preserved.

Each window size is also tested at multiple offsets but must fulfil the constraint that the window contains the frame with the WCET and only the most similar window at each window size is preserved at the end of the process.

The procedure described in algorithm 1 scales well as the number of frames increases. This can be informally seen from the following:

1. In the worst case, the pos_longest function will have to scan every frame's encoding time to find the frame which had the longest encoding time (which is then stored in WC_p). Thus it has a complexity bound of $O(frame.size)$.

2. The while loop on lines 3–16 is bound by the number of steps required to reduce $frames.size$ until it is less than or equal to 100, where each step reduces the current window size to 85% of its previous value. This can be seen as a similar problem to the calculation of compound interest. By rearranging the equation $frame.size(0.85)^n = 100$ an algorithmic bound of $O(\log(frames.size))$ was found.

3. The for loop on lines 8–15 will only compare those windows that contain the frame with the longest encoding time. This constrains the loop upper complexity to $O(window.size)$ – its worst case is when the WC_p can be the first and last frames of the window without making the window exceed the first or last frames.

4. The EMD function on line 10 has a complexity of $O(window.size) + O(frame.size - window.size)$ times before producing a final value in the worst case.

Combining the above and only keeping the terms contributing the most gives a total complexity of $O(n(\log n)^2)$ where n is the total number of frames thus this method is considered to be scalable.

4. Evaluation

The evaluation is split into two parts; understanding the dependency on the parameters and then how to

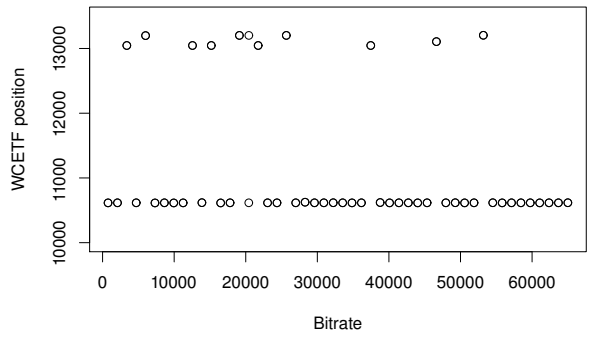

Figure 1. Graph comparing bitrate parameter to WCET frame position

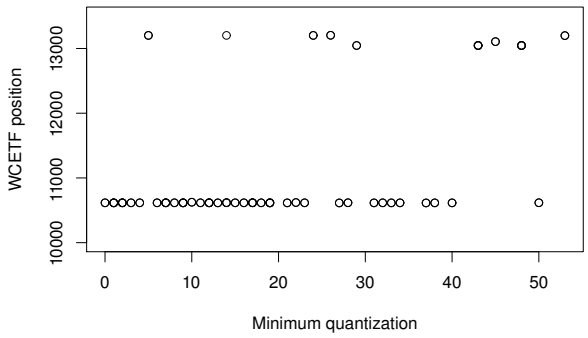

Figure 2. Graph comparing minimum quantization parameter to WCET frame position

find the smallest subset of the original input stream. The following subsections deal with these in turn.

4.1. Configuration Parameter Dependency

A test was performed to identify whether the location of the observed Worst Case Execution Time Frame (WCETF) was dependent on the encoding configuration parameters. To achieve this, graphs and tables were created comparing each parameter against the WCETF position.

These results are shown in figures 1, 2, 3 and table 3. The results show all WCETF positions fall into one of two groups - the first being the positions from 10614–10625 and the second 13046–13204. The lower group has the largest frequency with 79% of all results falling within it. Figures 1 and 2 show a spread of WCETF positions across bitrates and minimum quantization levels with no clear correlation between high and low positions. Table 3 shows that the distribution of WCETF position frequencies when error resilience is off is similar to the distribution when it is on. Further, both distributions are similar to those obtained when analysing those encoding parameter sets where frame scaling was enabled or disabled. As with bitrate and minimum quantization, there is no clear correlation.

Only the maximum quantization in figure 3 shows a trend, where all the high group WCETF position samples occur when the maximum quantization is 30 or above. However, a high maximum quantization does not prevent WCETF positions from the lower group – the figure shows a spread of low WCETF positions no matter what the maximum quantization setting is.

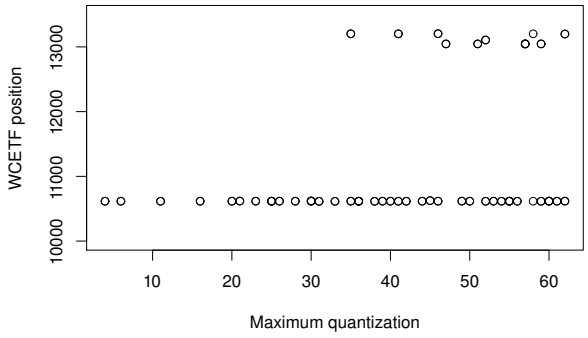

Figure 3. Graph comparing the maximum quantization parameter to WCET frame position

Due to these two distinct groups of WCETF positions, there is a need to validate a window's performance across multiple parameter sets because a window whose end only just captures the lower WCETF positions may not be suitable for windows of 20% or smaller.

4.2. Finding a Robust Subset of the Input Stream

By converting all possible parameter sets into a unique number, 100 parameter sets were chosen at evenly spaced intervals throughout the input space with

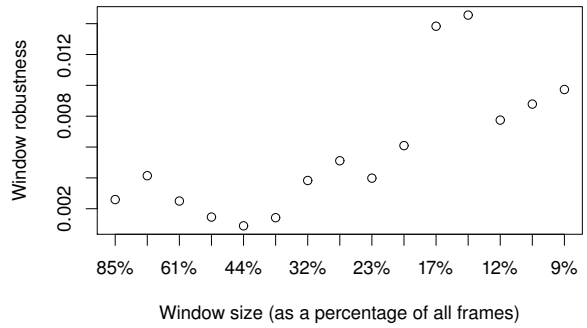

Figure 4. Graph comparing window size to its robustness using all frames within the window

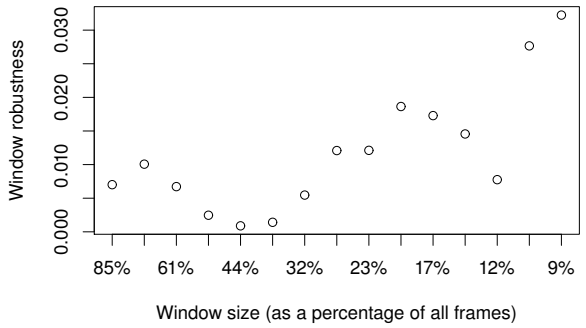

Figure 5. Graph comparing window size to its robustness using only the upper quantile of of frames within the window

the offset being chosen randomly.

To compare performance across multiple parameter sets, the best performing windows of each parameter set (as found using the algorithm 1) were reused with all the other parameter sets. For example given two parameter sets A and B, if parameter set A produced a frequency distribution F_a and W_a defined the frames in the best fitting subset window, the evaluation would run an EMD test to compare the cycle time frequency distribution of F_b to the distribution of frames in W_a.

In figure 4, good robustness is defined as being a window which fits distributions generated by all encoding parameters well (returns low EMD scores) and the scores returned do not differ by a large amount. Conversely, a window that returns high EMD scores across all parameter sets and returns scores that vary by large amounts will have a robustness value closer to 100.

As window sizes are reduced, robustness actually improves before becoming worse. This suggests large windows struggle due to the large number of frames being captured leading to the situation where a good fit is biased towards the start of video due to the need to collect many samples. The bounded fluctuation continues until a window size of 20% is reached which is significant as it is no longer possible to span both WCETF position groups with a single window.

Within the distributions, less than 1% of samples fall into the highest five bins making the most attractive windows those that have a similar spread of low cycle frames to the total distribution at the expense of higher cycle frame times. To counteract this bias, additional comparisons were performed only using samples from upper quantiles of the subsets. These results are reflected in figure 5, which shows a similar pattern of fluctuation extending down to window sizes of 12%. These results show that it is possible to reduce the number of frames encoded in a video while still maintaining the WCET attribute.

4.3. Validation with Alternative Video

To show the technique was not limited to the test video, the subset reduction was also applied to the one minute Paris video from the ITU-T video encoding conditions standard [16]. This video consists of 1065 frames of two presenters talking to the camera. Both the original Big Buck Bunny video and the Paris video were encoded with the 100 encoding parameters specified in tables 1 and 2.

The first of the two experiments compares the frequency distribution of all the frames within the Big Buck Bunny video to that of all the frames in the Paris video. Across all different encodings the mean EMD score is 0.018 and a standard deviation of 0.002. This demonstrates that frame encoding times are found in similar quantities regardless of the video. Since this is the case, it allows us to determine that the encoding process will produce distributions that do not change when the input video is long enough.

The second experiment analyses the performance of a smaller window. The distribution of the 20% window containing frames 8976–11794 for Big Buck Bunny was compared to the distribution generated by all the frames the Paris video. The window selected was based on results obtained from the upper 10% quantile

of frames when sorting by encoding time. As with the previous test the mean EMD score is 0.018 and the standard deviation is 0.002. Thus insights obtained from the subset of one video apply to all other videos because performing the subsetting on them produces a similar result.

These comparisons demonstrate that the reduced video sample is representative of multiple videos regardless of how encoding parameters are changed. This is because the distributions are declared to be similar with low EMD scores. Further, this score is achieved with most encoding parameters as evidenced by the low standard deviation.

5. Conclusion

This work introduced the problem of identifying the WCET for programs with a large input space. An example search of the input space for a video encoded with the VP8 codec was explored and a statistically based method for reducing the input space was tested. The experimental results showed that the input size could be reduced to 20% when considering all frames or 12% if considering only those longer execution frames for the distribution comparison. Using different video, a similar set of results were demonstrated showing that results obtained from one subsetted video are still applicable to other videos.

The results demonstrate that input space reduction before the application of existing WCET analysis allows analysis of problems previously thought too large or complicated. In addition to being able to attack new problems, existing problems which are amenable to reduction can be solved faster allowing fewer resources to be expended during the analysis stage or allowing a more thorough analysis to take place. Future work will extend this technique to other scenarios to test its applicability to different types of software.

References

[1] A. Betts, G. Bernat, R. Kirner, P. Puschner, and I. Wenzel. WCET coverage for pipelines. Technical report, Techincal report for the ARTIST2 Network of Excellence, 2006.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[3] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of the ACM/IEEE conference on Supercomputing*, Supercomputing '00, 2000.

[4] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. 4th edition, 2009.

[5] Canalys. `http://www.canalys.com/pr/2011/r2011013.html`, 2011.

[6] J. Chilenski and S. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193 –200, 1994.

[7] A. Colin and S. M. Petters. Experimental evaluation of code properties for WCET analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, pages 190–199, 2003.

[8] P. Emberson and I. Bate. Stressing search with scenarios for flexible solutions to real-time task allocation problems. *IEEE Transactions on Software Engineering*, pages 704–718, 2010.

[9] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks - past, present and future. In *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis*, 2010.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. *IEEE International Workshop on Workload Characterization*, pages 3–14, 2001.

[11] U. Khan and I. Bate. WCET analysis of modern processors using multi-criteria optimisation. *Empirical Software Engineering*, pages 5–28, 2010.

[12] E. Levina and P. Bickel. The Earth Mover's distance is the Mallows distance: some insights from statistics. In *Proceedings of the 8th IEEE International Conference on Computer Vision*, volume 2, pages 251–256, 2001.

[13] S. Mohan and F. Mueller. Hybrid timing analysis of modern processor pipelines via hardware/software interactions. In *Real-Time and Embedded Technology and Applications Symposium*, pages 285–294, 2008.

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

[15] J. Puzicha, J. Buhmann, Y. Rubner, and C. Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, volume 2, pages 1165–1172, 1999.

[16] G. Sullivan and G. Bjontegaard. Recommended simulation common conditions for H.26L coding efficiency experiments on low-resolution progressive-scan source material. *ITU-T SG16 Q. 6 (VCEG)*, 2001.

[17] J. Tate and I. Bate. Sensornet protocol tuning using principled engineering methods. *The Computer Journal*, 53(7):991–1019, 2010.

[18] N. J. Tracey. *A Search-Based Automated Test-Data Generation Framework for Safety Critical Software*. PhD thesis, University of York, 2000.

[19] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. System noise, OS clock ticks, and fine-grained parallel applications. In *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, pages 303–312, 2005.

[20] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6:127–135, 1997.

[21] 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. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, 2008.

ER	BR	MINQ	MAXQ	FRS
Off	12555	43	47	Off
Off	12564	43	47	On
Off	15184	48	57	On
Off	15193	48	57	Off
Off	17804	4	53	Off
Off	17813	4	53	On
Off	20424	14	58	Off
Off	20433	14	58	On
Off	2075	7	23	Off
Off	2084	7	23	On
Off	23044	0	25	Off
Off	23053	0	25	On
Off	25664	5	35	Off
Off	25673	5	35	On
Off	28284	10	45	Off
Off	28293	10	45	On
Off	30904	15	55	Off
Off	30913	15	55	On
Off	33533	2	20	Off
Off	33542	2	20	On
Off	36153	12	25	Off
Off	36162	12	25	On
Off	38773	22	30	Off
Off	38782	22	30	On
Off	41393	32	35	Off
Off	41402	32	35	On
Off	44013	40	42	Off
Off	44022	40	42	On
Off	46633	45	52	Off
Off	46642	45	52	On
Off	4695	17	28	Off
Off	4704	17	28	On
Off	49262	50	62	On
Off	49271	50	62	Off
Off	51882	9	55	Off
Off	51891	9	55	On
Off	54502	19	60	Off
Off	54511	19	60	On
Off	57122	2	30	Off
Off	57131	2	30	On
Off	59742	7	40	Off
Off	59751	7	40	On
Off	62362	12	50	Off
Off	62371	12	50	On
Off	64991	17	60	On
Off	65000	17	60	Off
Off	7315	27	33	Off
Off	7324	27	33	On
Off	9935	37	38	Off
Off	9944	37	38	On

Table 1. Tested parameter sets

ER	BR	MINQ	MAXQ	FRS
On	11249	9	11	Off
On	11258	9	11	On
On	13869	14	21	Off
On	13878	14	21	On
On	16489	19	31	Off
On	16498	19	31	On
On	19109	24	41	Off
On	19118	24	41	On
On	21729	29	51	Off
On	21738	29	51	On
On	24358	34	61	On
On	24367	34	61	Off
On	26978	8	39	Off
On	26987	8	39	On
On	29598	18	44	Off
On	29607	18	44	On
On	32218	28	49	Off
On	32227	28	49	On
On	3380	43	57	Off
On	3389	43	57	On
On	34838	38	54	Off
On	34847	38	54	On
On	37458	48	59	Off
On	37467	48	59	On
On	40087	1	59	On
On	40096	1	59	Off
On	42707	6	6	Off
On	42716	6	6	On
On	45327	11	16	Off
On	45336	11	16	On
On	47947	16	26	Off
On	47956	16	26	On
On	50567	21	36	Off
On	50576	21	36	On
On	53187	26	46	Off
On	53196	26	46	On
On	55816	31	56	On
On	55825	31	56	Off
On	58436	3	36	Off
On	58445	3	36	On
On	6000	53	62	Off
On	6009	53	62	On
On	61056	13	41	Off
On	61065	13	41	On
On	63676	23	46	Off
On	63685	23	46	On
On	760	33	52	Off
On	769	33	52	On
On	8629	1	4	Off
On	8638	1	4	On

Table 2. Tested parameter sets

WCETF	ER Off	ER On	FRS Off	FRS On
10614	2	4	3	3
10615	15	2	8	9
10616	17	25	22	20
10617	2	6	4	4
10618	2	0	1	1
10620	0	2	1	1
10625	0	2	1	1
13046	6	4	5	5
13106	0	2	1	1
13199	2	0	1	1
13202	2	3	2	3
13204	2	0	1	1

Table 3. Longest frame locations frequencies with differing error resilience and frame scaling settings