Entropic Complexity Measured in Context Switching

Paul Pukite and Steven Bankes
BAE Systems
USA

1. Introduction

Abstract: A complexity metric for concurrent software controlled systems is defined and derived. It is equivalent or comparable to the Shannon information metric, which essentially measures entropy of a system, but uses a novel and efficient technique based on a FFT to calculate its value. This can be extended to other temporal realizations of behaviour.

For concurrent software, the amount of context switching that occurs indicates the level of complexity of the program. If the program consists of a fixed period cyclic scheduler, context switches will occur very predictably at those same fixed intervals. However, if the program consists of a mix of periods, some long, some short, and some perhaps sporadically aperiodic, the complexity of such a program will be much greater. Further, the greater the spread between short periods and long periods will indicate that the program will be much harder to verify, as the shorter cycles will accumulate more testing coverage than the longer or sporadic interval context switching.

In some sense, this is what makes clock-driven synchronous logic much easier to test. The state space of possible events is reduced in direct relation to the reduction of available slots for computation. As the context switching and potential interactions between threads occurs only at these slots, a simplification of the temporal behaviour here will result in a better chance to verify the correctness of its execution.

By the same token, any concurrent program will show greater non-determinism than an equivalent sequential program. The benefits of making a concurrent program more synchronous grants a greater predictability in its execution semantics. In terms of meeting hard real-time constraints, a synchronously designed program will have predictable points for schedulability, allowing for techniques such as rate-monotonic scheduling (Klein, 1993) to meet strict timing deadlines. These categories of techniques grant a concurrent program the same possibilities for verification as a sequential one.

Even though these strict scheduling programs are effective for their problem domains, they are difficult to maintain and do not scale that well for the large software systems required. In fact most interactive programs and other soft real-time systems use the concept of event-driven semantics, which can allow interruption at any point in time. These have the benefit of dealing with interactions only upon demand, and so scale better, especially in terms of not placing a huge computational or communication load on the system when not needed.

New events can also be handled well, either by adding a new thread or task or interrupt to watch for their occurrence or by adding an event handler to a queue.
Unfortunately these benefits of scalability and flexibility detract from being able to reason about the execution semantics of a program, thus leading to an element of non-determinism. One can thus consider the obvious potential for introducing stochastic ideas to qualitatively and quantitatively understand how complex a design has become. If we can attach a simple and fundamental measure to this complexity and thus generate a useful metric, it has the capacity for comparing software designs or of providing a continuous benchmark of growing potential complexity. That leads to the notion of supplying an entropy-based measure to characterize the behaviour of a concurrent software system.

2. Approach – The context-switching entropy measure

The premise is to evaluate the complexity of a concurrently executing program. A novel approach of creating such a complexity metric involves the analysis of the context-switching frequency spectrum. We take a Fourier transform (*FT*) of the temporally distributed context switching events, *c(t),* and treat that as a probability density function (Feller, 1957) in frequency space (i.e. a normalized power spectrum).

$$p(f) = \left|FT\big(c(t)\big)\right| = \left|\int_{-\infty}^{\infty} c(t) \cdot e^{-i2\pi f t}\, dt\right| \tag{1}$$

Then the Shannon information entropy (*S*) of *p(f)* (Leon-Garcia, 2008) will generate a simple complexity measure.

$$S = -\int_{0}^{\infty} p(f) \cdot \ln\big(p(f)\big)\, df \tag{2}$$

The entropy of a spectrum has meaning because the power spectrum itself transforms the autocorrelation of the signal in the time domain. The autocorrelation is nothing more than the probability of a time-shift occurring between points of interest. Thus by transforming the time domain information to the frequency domain we do not lose the essential information pertaining to order and disorder in the signal, in particular when it comes to comparing one waveform against another. The information containing the essential order is largely retained across the transformation.

$$prob(t) \approx autocorrelation\big(c(t)\big) = \int_{0}^{\infty} c(t) \cdot c(t+\tau)\, d\tau \tag{3}$$

So the Fourier transform represents the same probabilities as that of the autocorrelation but computed in the frequency domain. For any captured sequence of events, the Fourier transform of the autocorrelation is practically obtained as the magnitude of the Fourier transform of the waveform.

$$p(f) = FT\big(prob(t)\big) = |FT(c(t))| \tag{4}$$

For a given mean frequency <*f*>, the value of *S* will be the greatest when the probability spread of the context switching frequency components is maximized. In this case, complexity tracks entropy and the maximum entropy principle (Jaynes, 2003) can be applied to understand the trend toward greater complexity. By Parseval's theorem, we do not lose the integrated signal when we take the Fourier transform.
This general approach is also known as Frequency Domain Entropy (Bao, 2004).

	Signal	Entropy	FT(Signal)	Entropy
Periodic Impulse Train	$\sum_{n=-\infty}^{\infty} \delta(t-nT)$	low	$\frac{2\pi}{T}\sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{2\pi}{T}\right)$	low
Disordered Autocorrelation with Mean	$\alpha \cdot e^{-\alpha t}$	1+ln(α)	$\frac{\alpha}{\alpha^2+\omega^2}$	~ 2+ln(α)
Noise	Random(u(t))	Maximum entropy over time interval	Random in frequency domain	Maximum entropy over frequency range

Table 1. Mapping of signal entropies to their corresponding frequency domain entropies. The trend of increasing entropies with increasing disorder is maintained through the transformation process.

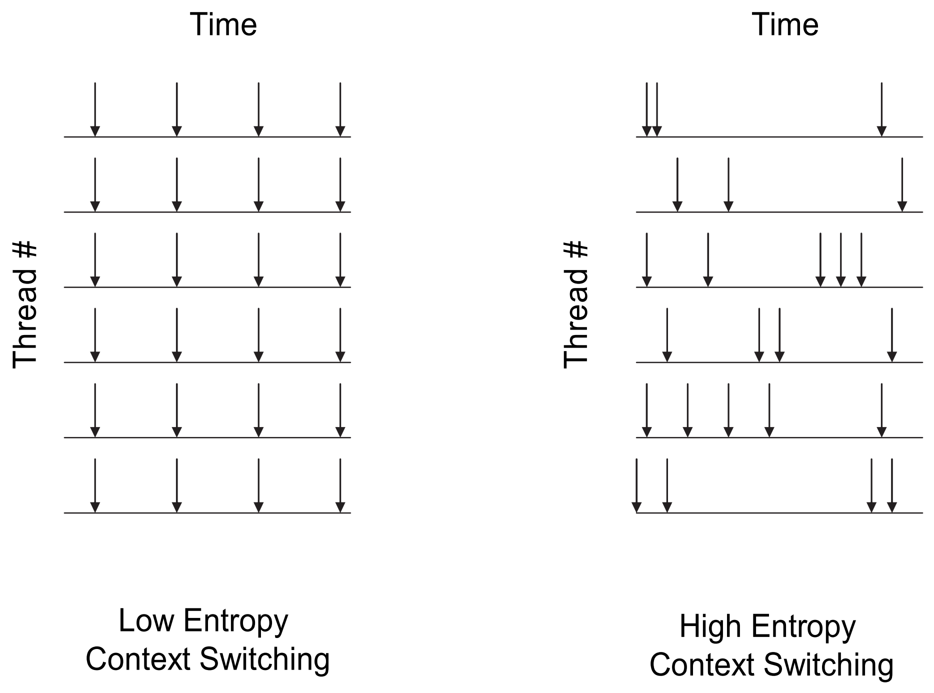

Fig. 1. Example of context-switching waveforms with low entropy (left) and high entropy (right). The low entropy signals are more regular.

In practice using a discrete Fourier transform such as the Fast Fourier Transform (FFT) (Brigham & Morrow, 1967) allows us to then use the summation variant of the entropy, where N is the order of the FFT.

$$S = -\sum_{i=1}^{N} p(f_i) \cdot \ln\left(p(f_i)\right) \tag{5}$$

The utility of the frequency domain approach is apparent when we consider that the frequency domain is partitioned into FFT bins and we have a very fast way of computing the metric, and in many cases more convenient than by calculating the autocorrelation, or by traditional periodogram (Schuster, 1898). If we did have the extra processing time, it may make sense to calculate the entropy off the autocorrelation as that would definitely retain more of the information measure, and thus not losing information due to the transformation process. In practice, however, many times we want to know only *relative* entropy measures, so the loss in precision is not as important as long as we maintain the relative ordering or rank.

As an intuitive interpretation of this measure consider that a value of S near 0 (if $<f>$ normalized to 1) indicates that the program is very specialized and is only trying to a few things at once. As S approaches 1 it becomes a general purpose application that attempts to control many different behaviours or responds to events of a more random nature. If S goes much greater than 1, then the temporal complexity turns even more random, even approaching fractal. It becomes progressively harder to test such systems because of the diverging spread in time scales.

This turns into a useful metric since any sufficiently capable controller program or event-driven system will try to accommodate as much real-world functionality as possible, and since the real world tends to disorder and to maximize entropy, this will track an increasingly complex scenario. So in practice, we will have fast cycles interspersed with slower cycles corresponding perhaps to human events or sporadic signals.

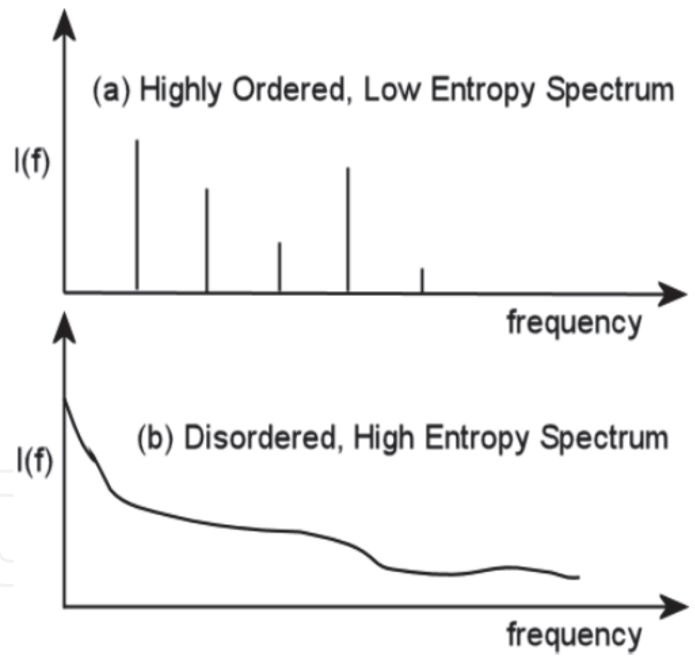

Fig. 2. Variation of power spectrum with increasing disorder. (a) Ordered signal with harmonics generates a low entropy signal. (b) Disordered signal generates a larger intropy.

In further practical terms, to apply the metric we need to run a simulation or collect data from a real or prototype system to gather the statistics. It may be possible to evaluate a program statically but that would require an introspective analysis of a program's structure, a task much more involved than collecting the statistics during typical use or as a test scenario.

The possibility of using this approach on non-concurrent programs such as a single-tasking event-driven program exists but these have less inherent complexity due to their sequential nature. When the concurrency becomes an integral aspect of the system and in particular when threads must interact with one another does the complexity start to increase quickly. That becomes the essence of the disorder or entropy that we try to capture.

System designers will use tools such as state diagrams, activity diagrams, and Petri nets (Peterson 1981) to architect concurrency behaviors. A typical Petri net is shown below which illustrates the parallel activities and synchronous behaviors. These have various levels of contol, in that they can abstract some of the temporal scale differences by encapsulation of behaviours (Pukite & Ludwig, 2007). However, the actual synchronization will always reveal itself in the realized system.

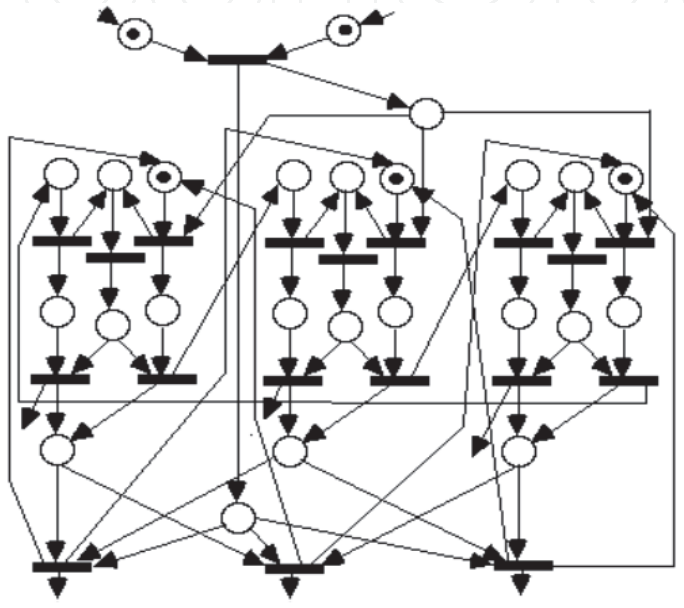

Fig. 3. A typical concurrent automata represented by a Petri net. Multiple threads are represented by filled tokens (places), and context switches occur when threads meet at synchronization points (bars). For large systems, the complexity of interactions canl grow well beyond this level.

2.1 Application

To effectively put the temporal complexity measure into practice, the program or simulation implementation will need a way to time stamp and data log context switches during execution. An implementation of a discrete event task scheduler such as the public source Degas (Ludwig & Pukite, 2006) provides such an instrumentation facility.

The simplest input stream is a list of events signified by time-stamps, which will serve to capture the *c(t)* signal. The values for *c(t)* can be as simple as delta functions with a value of unity indicating a context switch.

2.2 Computation

This algorithm needs a FFT routine. The following pseudo-code guides the calculation calling a generic FFT routine named **Fourier8**. This assumes that each line contains the number of events that occur in the time step and is read through standard input. (This is set to read in exactly 2N lines, where N is an order 15 FFT) If sparse input is provided with time stamps then the data must be transferred into a discrete set of lines with either 0 or N events on each line, due to the nature of the discrete time FFT.

```
with Fourier8;
with Text_IO;
with Ada.Numerics.Generic_Elementary_Functions;
procedure Context_Switching_Entropy is
   subtype Real is Long_Float;
   package mth is new
     Ada.Numerics.Generic_Elementary_Functions (Real);
   use mth;
   Log_Of_Max_Data_Length : constant := 21;
   type Array_Index is range 0..2**(Log_Of_Max_Data_Length+1)-1;
   type Data_Array is array(Array_Index) of Real;
   package fft8 is new Fourier8
     (Real, Array_Index, Data_Array, Log_Of_Max_Data_Length);
   use fft8;
   D_Re, D_Im : Data_Array;
   N : Data_Index;
   Transformed_Data_Last : Data_Index;
   Exp_Table : Exp_Storage;
   Val : Real;
   Sum : Real := 0.0;
   Prob : Real;
   Entropy : Real := 0.0;
begin
    N := Data_Index (2**15 - 1);
    for I in 0..N loop
      -- Read from standard input
      Val := Real'Value(Text_IO.Get_Line);
      D_Re(I) := Val;
      D_Im(I) := 0.0;
   end loop;
    FFT
     (Data_Re                  => D_Re,
      Data_Im                  => D_Im,
      Transformed_Data_Last    => Transformed_Data_Last,
      Input_Data_Last          => N,
      Exp_Table                => Exp_Table,
      Inverse_FFT_Desired      => False,
      Normalized_Data_Desired  => True);
   for I in Data_Index range 1 .. N-1 loop
      Val := sqrt(D_Re(I)*D_Re(I) + D_Im(I)*D_Im(I));
      Sum := Sum + Val;
   end loop;
   for I in Data_Index range 1..N-1 loop
      Val := sqrt(D_Re(I)*D_Re(I) + D_Im(I)*D_Im(I));
      Prob := Val / Sum;
      if Prob /= 0.0 then
         Entropy:= Entropy - Prob * Log(Prob, 2.0);
      end if;
   end loop;
   Text_IO.Put_Line ("Entropy =" & Entropy'Img);
 end Context_Switching_Entropy;
```

Fig. 4. Pseudo-code algorithm for calculating the context-switching entropy using a library package FFT routine

Note that for a running simulator, if the data is provided continuously through an output port or pipe, then a suitably fast FFT can report a context switching metric in real time. This is useful for diagnostics, for example correlating the detection of errors with high complexity regimes.

2.3 Example of use

We evaluated hypothetical systems with 15 concurrent tasks. One system featured asynchronous events. The other system had tasks cyclically scheduled at different periods. The asynchronous system always had higher switching entropy. The context switching metric distinguishes complicated but synchronized architectures from those with complex temporal behaviour

For this metric, by computing the entropy of the phase spectrum in addition to the amplitude may help in discriminating between complexity and noise. That would be straightforward to include because right now we discard the phase information. Any stationary signal that shows asymmetry must have some peculiar phase relationships going on. So it might be easier to discount randomness in favour of more complex phase relationships if we were to include both the amplitude spectrum and the phase spectrum in the final metric.

Another simple alternative that works on the timing alone is the **gzip** program. This looks at the distribution complexity in times and calculates the entropy metric. The usage of this application is trivial. Take the output file, if a sequence of discrete inputs (zeros and number of values), run it through **gzip** and record the size (Benedetto, et al, 2003). Comparisons of two different sequences of identical length will suggest that the less complex program is the one of the smaller zipped size.

3. Complementary approach – the multi-scale entropy measure

An alternate approach to measuring the temporal complexity involves the application of the multi-scale entropy metric suggested by (Costa, 2005). This differs from the just described context-witching metric in that it measures the complexity of a temporal behaviour or signal over a wide range of fundamental periods. Whereas the single-scale metric works best over a single-decade frequency spectrum scale, a multi-scale metric offers up the possibility of looking at complexities at a variety of time scales, ranging over potentially orders of magnitude. This is definitely useful but it can't be boiled down into a single complexity metric; instead we will need to depict this graphically over what amounts to a logarithmic frequency scale.

The basis for the multi-scale entropy metric is that many real-world behaviours often occur over time scales of varying dynamic ranges. Costa originally applied this to a biomedical application, trying to extract the temporal complexities of cardiac-driven circulatory systems. When such a pulsed cardiac signal is multi-scale, it is actually composed of a fundamental pulse and various arrhythmias, leading to a complicated spectrum of events. The signal actually appears buried amongst competing behavioural periodicities at different time scales and so it becomes that much harder to extract the information.

At first glance, we would imagine that a Fourier transform would work well to extract the periods but in fact a typical FFT algorithm actually works best over a limited dynamic range. By expanding the scope to a multi-scale level, Costa showed that this complexity measure has use in a real-world application and we contend that it may also prove useful for cyber-physical applications such as a complex event-driven system. In this case, the time scales can range from fast interrupt-processing, to human-scale interactivity, to the even more sporadic environmental influences.

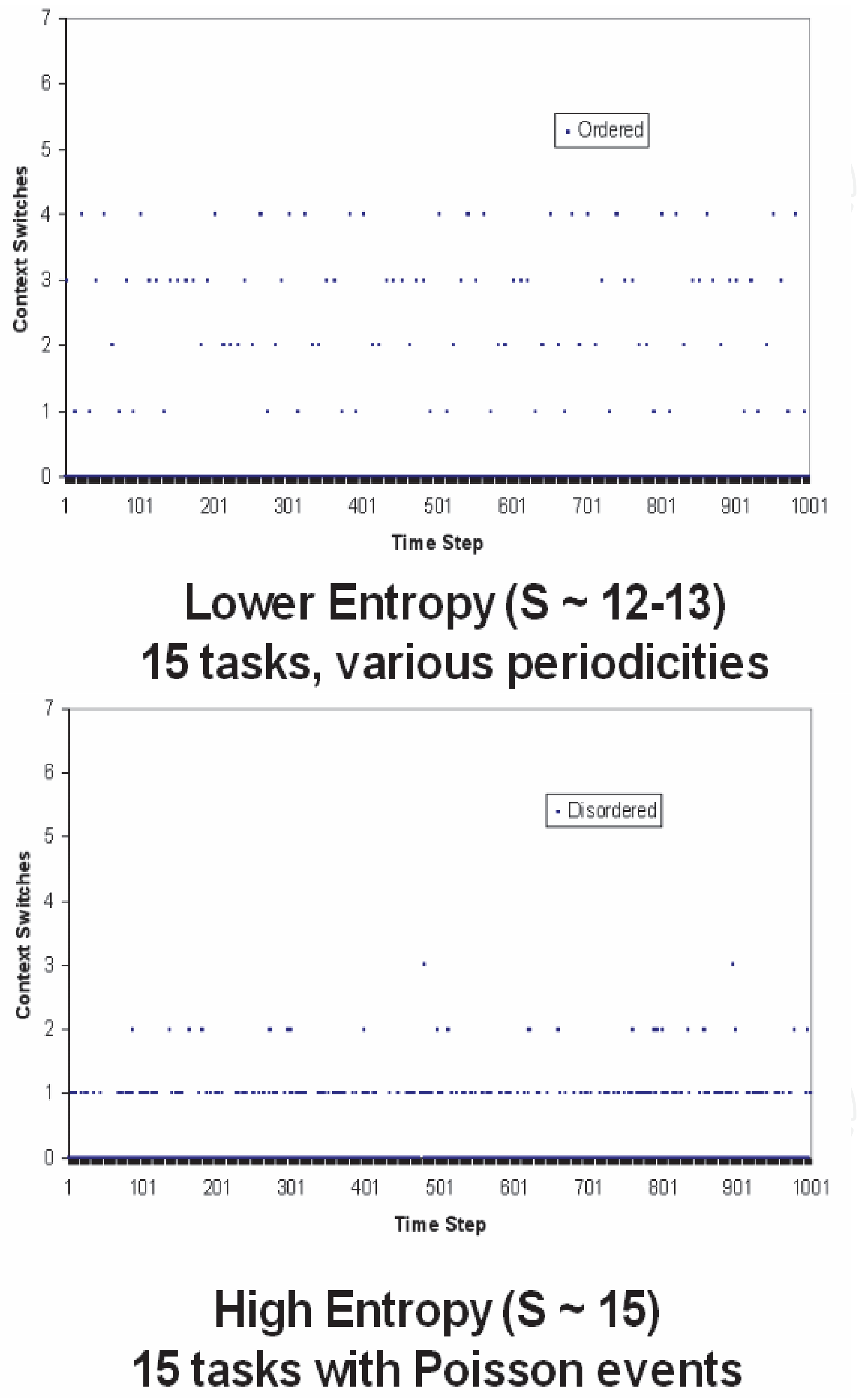

Fig. 5. Context switching metric distinguishes complicated but synchronized architectures from those with complex temporal behaviour.
(top) High Entropy (S ~ 15) 15 tasks with Poisson events
(bottom) Lower Entropy (S ~ 12-13) 15 tasks, randomized events

The multi-scale aspect senses the different temporal frequencies in the underlying signal, comparable to what the contest-switching metric does, but instead pairs or groups the data points to measure a different coarse graining effect. This works out fairly straightforwardly in terms of a autocorrelation. The essential algorithm groups adjacent time samples together in a window of length *Scale* as the coarse graining or moving average measure. Then it counts the number of times, n, that the amplitude will change from one coarse-grained time step to the next.

If the amplitudes don't change for a given coarse-grain then it is predictable and the entropy will be low. To calculate the sample entropy they calculate

$$complexity(Scale) = -\log\left(\frac{n(Scale+1)}{n(Scale)}\right) \qquad (6)$$

over each of the scale factors *Scale* = 1 .. *maxScaleFactor*.

3.1 Usage domains

A graph of the multi-scale entropy will appear flat if it is measuring "1/f" (van der Ziel, 1950) or the so-called pink noise as the underlying behaviour. Pink noise shows a predictable constant change of amplitude density per scale factor; in other words it has a constant energy per frequency doubling while white noise shows constant per frequency interval (Montroll & Shlesinger, 2002).

In comparison to the structure-less noise, if structure does exist in the signal, you will see observable changes in the entropy from one scale factor to the next. For example a superimposed sine wave would show a spike downward in sample entropy when it crossed a harmonic in the scale factor.

A simple interpretation suggests that we scale the measured results relative to the *1/f* noise part of the signal. The *1/f* noise includes the greatest variety of frequencies of any behaviour known and therefore the highest entropy (Milotti, 2002). So by providing a good visualization or graph that plots the *1/f* asymptotic value we can immediately gauge the complexity of a signal. Costa *et al* discuss the difficulty of distinguishing between randomness and increasing complexity, which has importance in the realm of event-driven systems.

"*In fact, entropy-based metrics are maximized for random sequences, although it is generally accepted that both perfectly ordered and maximally disordered systems possess no complex structures. A meaningful physiologic complexity measure, therefore, should vanish for these two extreme states.*"

This is a key insight and one echoed by researchers in complexity theory (Gell-Mann, 1994) in that the most interesting and challenging complexity measures occupy the middle range of the complexity scale. In other words, the most ordered signals can be described by a few harmonic periods and at the other extreme, the complexity reduces to simple stochastic measures akin to statistical mechanics (Reif, 1965). In between these extremes, we require a different level of sophistication.

3.2 Comparison to single-scale metric

The same inputs as that for the Context-Switching Metric described earlier result in the following *Figure 6*. Note that in this case as well, the sample entropy is always higher for the disordered signal than for the ordered signal. The reference *1/f* noise level is shown on the plot to indicate the asymptotic maximum entropy level achievable.

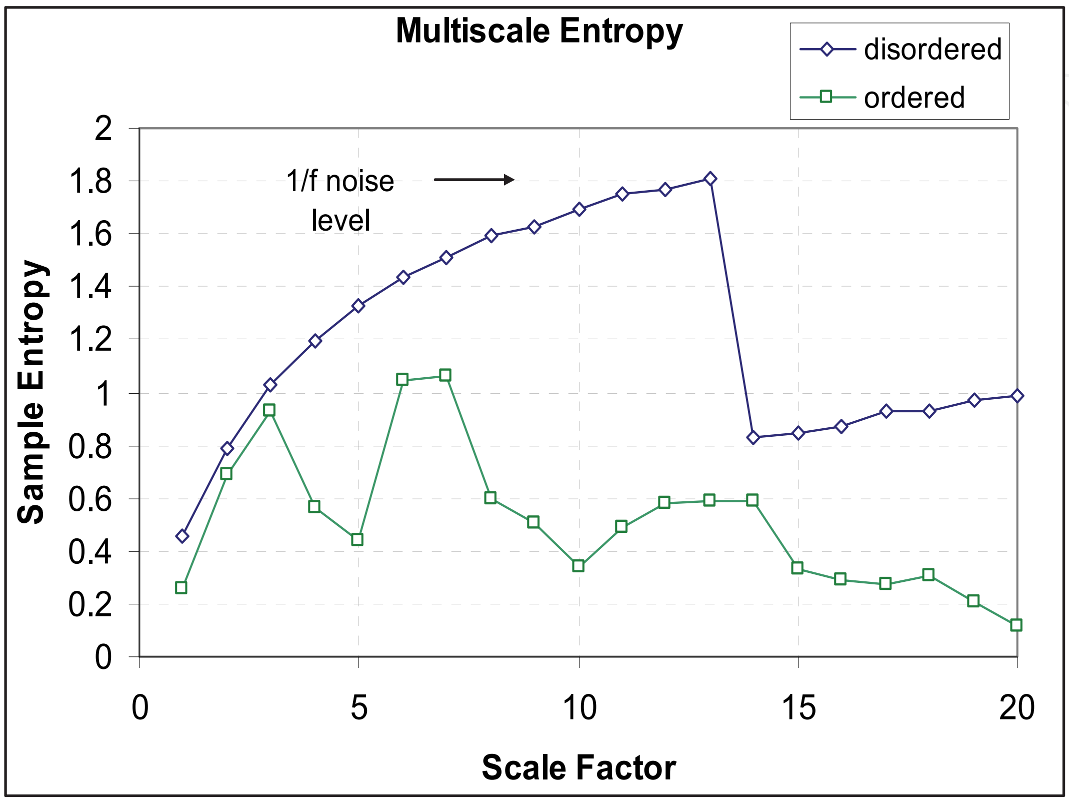

Fig. 6. For the same pair of inputs we used on the context-switching metric, the multiscale entropy appears as the following graph. It shows greater variety than the context-switching metric over the time scales because the metric compares at different levels of resolution.

In practice, the multi-scale algorithm requires only a basic periodogram method invoked over different time scales. The output is one value per temporal scale factor so the results are best displayed as a graph, via a spreadsheet or bar-charting software for example. The calculation is somewhat more brute force compared to the FFT, with complexity $o(n^2)$ versus $o(n*ln(n))$. The context-switching metric operates over a narrower time scale so gets rolled into a single value, simplifying the presentation into a classical scalar metric.

4. Discussion

Interacting concurrent activities can produce behaviour that is difficult to anticipate. The combinatorial capacities of just the ordering of parallel thread execution will exhaust any brute force attempt at testing the possible permutations. Moreover, the possibilities of rare anomalies attributed to corner cases of executions may take non-determinant times to reveal, or worse, are not easily repeatable. The sporadic defect that occurs rarely in the lab usually has a real origin and should never be ignored **(Kuhn, et al, 2004).**

The figure below shows a representative timeline trace of the interacting subsystems on a typical automated ground vehicle. The task interactions are interleaved and pipelined.

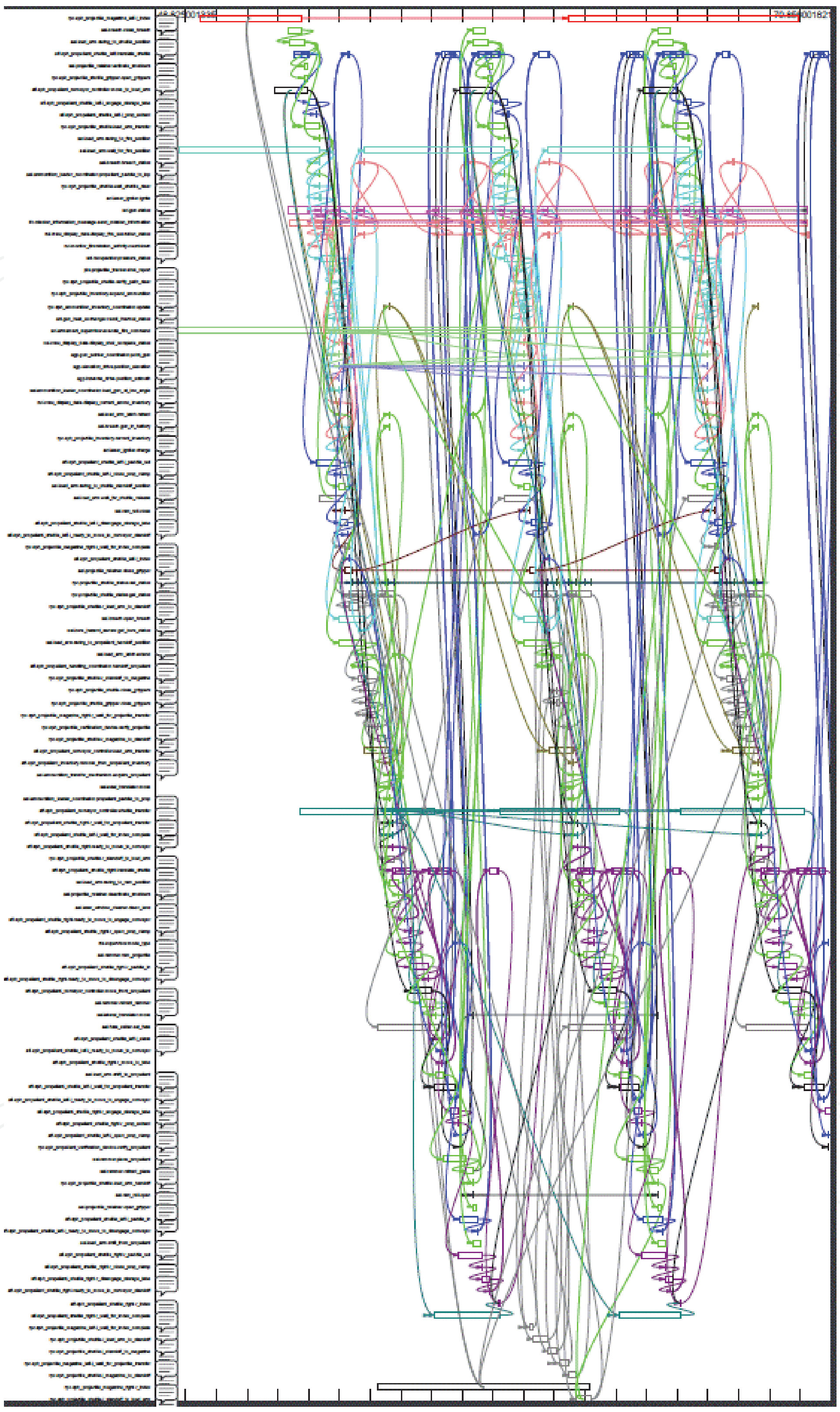

Fig. 7. GANTT chart of a typical vehicle system execution trace showing interacting threads. Time proceeds left to right, and one thread exists per horizontal entry. The lines indicate thread synchronization points. This diagram is only meant to give a notional idea of complexity, and the text description along the left edge is irrelevant to the discussion.

4.1 Applications to design

The difficulty in designing and debugging of such systems necessitates practical simulation tools and simplified metrics such as the entropy measures described here. Both of these entropy metrics are potentially useful during system development as an *analysis of alternatives* or *design space exploration* tool. If several concurrent options are available for a design, either of these metric could provide criteria to establish the least complex design. In that sense, it serves as a simple utility function to measure unwanted or creeping complexity much like a duty-cycle utilization measures processor contention.

4.2 Applications to test and verification

A close connection exists between the entropy metrics and usage modelling (Whittaker, 1994) for program verification. Most non-exhaustive testing requires a mix of tests taken during nominal conditions along with tests sampled according to potentially rare conditions. This consideration takes into account the number of test vectors and the path coverage for testing. Any characterization at this stage will provide useful inputs to generating a stochastic measure of reliability. This could incorporate stochastic usage models and a log-likelihood metric is often used to compare between two state space probabilities. Between an entropic measure and a usage model, we can cover the temporal and path dimensions of a program's execution and its programmatic complexity.

4.3 Applications to diagnostics

As a diagnostic tool, the context switching metric can also detect potential complexities during execution. Since the FFT can easily compute in real-time for typical sample sizes N, parallel execution of the entropy algorithm with the context switching data can reveal deviations from expected operation. For example, if an execution profile shows a high regularity of frequent context switches during some interval and then transitions to a more irregular sequence of switches with the same overall density, the expected entropy measure will definitely increase. In that sense, the entropy metric measures an intrinsic property of the signal, and that strictly speaking, density fluctuations such as expected increases in the rate of context switches will not influence the measure. In other words, density alone does not affect the complexity.

By the same token, the multi-scale metric has obvious benefit for detecting long term complexity changes or short-term bursts buried in a nominally sampled signal. The idea of using frequency domain entropy for diagnostics of complex machinery is further explored in (Shen, 2000).

5. Conclusion

We described a complexity metric for concurrent software controlled systems or concurrent realizations of behaviour. The novel approach of creating a complexity metric for context switching involves the analysis of the switching frequency spectrum. We take a Fourier transform of the temporally distributed context switching events ($c(t)$) and treat that as a probability density function in frequency space (i.e. a normalized power spectrum). Then the entropy (S) of $p(f)$ will generate a simple complexity measure.

The context switching metric can be used during system development as an analysis of alternative utility function. If several design options or algorithms are available, the context-

switching metric can be used as selection criteria to minimize inherent algorithmic complexity. It is comparable or equivalent to the Shannon information metric, which essentially measures entropy of a system.
As an alternative approach we compared this against a multi-scale entropy measure. Although more involved in construction, the multi-scale entropy can be used as an orthogonal metric, perhaps more useful for measuring temporal behaviours of a wide dynamic range or as a more detailed diagnostic tool. This will reveal finer structures in complexity than the single-scale metric can.

6. Acknowledgment

We wish to thank the DARPA META program for providing encouragement to develop complexity metrics for cyber-physical system development.

7. References

Bao, L. Intille, S. S., (2004). Activity Recognition from User-Annotated Acceleration Data, Pervasive Computing, Lecture Notes in Computer Science, Vol.3001, pages 1-17, Springer-Verlag, ISSN 0302-9743

Benedetto, D. Caglioti, E. Loreto, V., (2002). Language Trees and Zipping, *Physical Review Letters*, Vol.88, No.4, ISSN 0031-9007.

Brigham, E. O. and Morrow, R. E., (1967). The Fast Fourier Transform, IEEE Spectrum, Vol.4, No.12, p.63-70, ISSN: 0018-9235 .

Costa, M., Goldberger, A.L., and Peng, C.-K. (2005). Multiscale entropy analysis of biological signals, *Physical Review E,* Vol.71, No.2.

Feller, W., (1957). *An Introduction to Probability Theory and Its Applications,* Volume I and II, John Wiley, 1957

Gell-Mann, M., (1994). *The Quark and the Jaguar: Adventures in the Simple and the Complex*, W.H. Freeman and Co, ISBN0-7167-2725-0.

Jaynes, E. T., (2003). *Probability Theory: The Logic of Science*, Cambridge University Press, ISBN 052159271.

Klein, M., (1993). *A Practitioner's handbook for real-time analysis: Guide to rate monotonic analysis for real-time systems,* Kluwer Academic Publishers, ISBN 0792393619 .

Kuhn, D.R., Wallace, D.R., Gallo, A.M., Jr., (2004). Software fault interactions and implications for software testing, *IEEE Transactions on Software Engineering*, Vol.30 No.6, p.418-421, ISSN: 0098-5589

Leon-Garcia, A., (2008). *Probability and Random Processes for Electrical Engineering*, Prentice-Hall, p.273, (2008).

Ludwig, L. and Pukite, P., (2006). DEGAS: discrete event Gnu advanced scheduler, *Proceedings of the 2006 annual ACM international conference on SIGAda*, ACM SIGAda Ada Letters, Vol XXVI Issue 3 , ISBN 1-59593-563-0.

Milotti, E., (2002). 1/f Noise: A Pedagogical Review, invited talk to E-GLEA-2, Buenos Aires, arXiv:physics/0204033v1

Montroll, E.W. and M.F. Shlesinger, (1982) .On 1/f noise and other distributions with long tails, Proceedings of the National Academy of Sciences, 79/10, 1982.

Peterson, J. L., (1981). Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ.

Pukite, P., and Ludwig, L., (2007). Generic discrete event simulations using *DEGAS*:: application to logic design and digital signal processing, *Proceedings of the 2007 ACM international conference on SIGAda,* ACM SIGAda Ada Letters, Vol XXVII Issue 3, ISBN 978-1-59593-876-3

Reif, F., (1965). *Statistical and Thermal Physics*, McGraw-Hill.

Schuster, A., (1898). On the investigation of hidden periodicities with application to a supposed 26 day period of meteorological phenomena, *Terrestrial Magnetism and Atmospheric Electricity*, Vol. 3, p.13-41.

Shen, L., Tay F.E.H., Qu. L., Shen Y., (2000). Fault diagnosis using Rough Sets Theory, *Computers in Industry,* Vol.43.

van der Ziel, A. (1950). On the noise spectra of semi-conductor noise and of flicker effect, 1950, Physica 16, 359-372.

Whittaker, M., (1994). A Markov chain model for statistical software testing, *IEEE Transactions on Software Engineering*.