Predictability as an Emergent Behaviour

Alan Burns and David Griffin
Department of Computer Science,
University of York, UK.
email {burns, djg}@cs.york.ac.uk

***Abstract*—One means of obtaining time predicable systems is to build them from predictable components which are themselves built from predictable subcomponents. But this is not the only approach. In this paper we illustrate the benefits that are possible if components are designed to exhibit independent random behaviour. The worst-case execution time of application code can be estimated (with low probability of failure) to be not much greater than what one would expect the average to be. Unfortunately this approach of obtaining predictability as an emergent property of the code's execution cannot be delivered by today's hardware. Means by which hardware can be made more accommodating to this objective are explored.**

I. Introduction

At the last workshop Lee [9] argued that predictable systems must be built from predictable components. Here we wish to explore an alternative framework in which components with more random behaviour are combined so that predictability (where it is needed) emerges. Predictability is a necessary property only at the application level; an application may have timing constraints that have to be adhered to, but how the necessary level of predictability is obtained is not usually prescribed. Ultimately all computer-based systems rely on the regular behaviour of electronic circuits that are composed of electrons with essentially random behaviour. The resulting (or emerging[1]) predictable behaviour comes from the enormous difference in scale there is between the system and its components; we consider here whether the 'effects of large numbers' can also be used for timing analysis.

In this paper we first explore, in a purely abstract way, the relation between randomly behaved components (at a low level of time granularity) and predictability (at a higher level of granularity). We then consider how hardware could be made to work in an appropriately random way.

The framework we employ to argue about systems that behave at different time scales is that of Timebands [2]. In the Timebands framework a system is assumed to consist not of a single time dimension but a finite set of partially ordered *bands*. Each band is represented by a *granularity* (expressed as a unit of time that has meaning within the band, e.g. the milliseconds band) and a *precision* that is a measure of the accuracy of the time frame defined by the band. System activities are placed in some band **B** if they engage in significant events at the time scale represented by **B**. They have dynamics that give rise to changes that are observable or meaningful in band **B**'s granularity. In real-time systems, applications usually have deadlines and periods in the millisecond range, whereas individual instructions on a fast modern embedded computer are measured in nanoseconds. This represents a scale difference of six orders of magnitude.

In the timebands framework a band is populated by *events* and *activities*. Events, in the normal way, are considered to be instantaneous (a cut of the time line) within the band of their definition. Activities by comparison have duration of one or more 'units' of the band's granularity. Within an activity there will be collections of events with precedence constraints defined to represent the behaviour of the activity. Bands are linked by mapping events in one band to activities in lower bands with finer granularity. For example an interrupt in the milliseconds band may be considered instantaneous; the actual interrupt handler will however have duration in the nanosecond band. The mapping between these two representations is constrained by the requirement for the activity in the nanosecond band to have a duration no longer than the precision of the millisecond band.

We have shown in previous work [3] how behaviour in an activity (failures in this case) can be represented by 'point' failures in a time band of coarser granularity. In this paper we are again interested in the behaviour at a coarser band (that applicable to the application), but behaviour that is derived from the collective behaviour of a large number of entities at a much lower time granularity. These entities do not need to be predictable, but their collective behaviour does. It is also clear that predications do not need to be absolute. Levels of imprecision are always allowed in any engineering process. Here we will use probability of the predication being false as the characteristic of imprecision.

II. Random Behaviour

Consider a completely abstract and theoretical machine in which the execution times of basic instructions are iid (independent and identically distributed). Let the execution time of all of the machine's instructions be either 1 or 10. This could perhaps represent the cost with cache hit or cache miss. Let the probability of a cache miss (execution time of 10) be 0.1 (ie. 10%). The average instruction execution time for the machine is therefore easily computed as 1.9.

Now we require for scheduling analysis the worst-case execution time, WCET, for a task whose longest path has

[1]Whether temporal predictability is actually an emergent property is a debateable point; there are many different definitions of emergence (see Stepney at al[12] for a discussion on possible definitions of emergence), certainly significant differences in level does appear to be one recurring characteristic.

	N=10^5	N=10^6	N=10^7	N=10^8
Average ET	190000	1900000	19000000	190000000
WCET, p=10^{-3}	192639	1908344	19026385	190083437
Average ET	190000	1900000	19000000	190000000
WCET, p=10^{-5}	193642	1911516	19036415	190115153
Average ET	190000	1900000	19000000	190000000
WCET, p=10^{-7}	194440	1914039	19044393	190140383
Average ET	190000	1900000	19000000	190000000
WCET, p=10^{-9}	195122	1916195	19051211	190161941

TABLE I
ESTIMATES BASED ON N AND P

been analysed to contain N instructions[2], say 100,000. For a standard definition of worst-case this leads to a value for the task of 100,000 x 10, ie. 1,000,000. The average being 190,000. But the absolute worst-case is not actually needed as there is a tolerance on the accuracy required from the timing analysis. One way of expressing this tolerance is via an estimate, E, of the worst-case execution time such that the probability of the execution time of an actual run of the task, A, being greater than E is less than some defined probability p. That is

$$prob(A > E) \leq p$$

So, for example, if N is 100,000 and our tolerance bound is 10^{-5} (probability, of any one execution time of the task having a value greater than the estimate, E being less than or equal to 10^{-5}) leads to a value of E of just 193,642. Note how close this is to the average value – it is approximately only 0.25% larger. Other values of E (for various combinations of N and p) are shown in Table 1. This table also includes the average execution time (Average ET) for comparison.

Within the timeband framework any temporal statement expressed within the context of a particular band is assumed to have an accuracy requirement given by the precision of the band. So a statement that the WCET of a component is C is true as long as the actual execution time is less than $C + \rho$ - where ρ is the band's precision. In general precision is no greater than 1% of granularity (of the band). It follow that in the examples given in this section WCET can be taken to be ACET (Average-Case Execution Time).

The values in Table 1 come the following straightforward analysis. Each of the N execution times is an independent random variable (S). For large N the actual execution time, A, of the sequence of instructions is the sum of N random variables and will be approximately Normally distributed by the Central Limit Theorem. The variance of A will be N times the variance of S. And the distribution [E-A]/sqrt[var(A)] has the standard N(0,1) normal distribution. From this one can compute the probability of any estimate E begin greater than A, or as we did above, compute E for a desirable probability bound p. Note Table 1 was actually produced directly rather than via standard tables as these don't usually go down to 10^{-5} level.

[2]Analysis can be undertaken on a set of paths, here we concentrate on just a single candidate for longest path.

Table 1 shows that for even very high levels of reliability, such as the probability of failure being less than 10^{-9} (which is sometimes quoted for the most stringent avionics applications), the estimate of WCET is very close to the average. This results from the size of N – the number of basic instructions in a path of the application's code. Here this could quite reasonable be 10^7 or more.

The above analysis has focused on the probability of failure of a single run of the task. More generally failure rates are expressed as a maximum per hour of execution. If we assume that our representative tasks has a period of 100ms then in an hour of execution it will execute 36,000 times. Taking again the program with 100,000 instructions and a required failure rate of 10^{-9}, then the execution time estimate for a single execution of the task is 195,122 (see Table 1). Now to compute the failure rate for a single execution (X) so that the cumulative failure rate per hour is 10^{-9} requires the solution of:

$$1 - (1 - X)^{36,000} = 10^{-9}.$$

This is solved (approximately) by the value $X = 10^{-13}$. Using the same analysis as before this delivers an estimate of of WCET of 196,275. Again a very small increase over the average.

If rather than our simple 1 or 10 execution time, our basic instruction times are more realistic but still iid then the above technique can still be applied. All one needs to be able to compute is the variance of the instruction's execution time.

Note this method of computing an adequate estimate of the worst-case execution time of a task does not involve the use of extreme value distributions such as Gumbel [8], [4], [7].

III. REAL/POTENTIAL HARDWARE

Current real hardware does not of course exhibit perfectly random execution times. But this does not mean that the idealized results form the above analysis cannot be adapted. Consider the following points.

- Instructions could be classified into M basic types, the worst-case path through a task would need to 'count' the occurrences of each basic type.
- Levels of dependence could be accounted for by the use of copulas [10], [1].
- Caches could be designed to exhibit more random behaviour - see work on random cache replacement policies from the PROARTIS Project [5].
- Other hardware components such as branch predication could be made more random.
- Shared buses on multi-core architectures could also be 'randonised'.

The latter point is an important one if we are to derive compositional analysis for multi-core architectures. Here we need the execution time of a task on one core to be independent of the details of the execution time behaviour of the tasks on other cores. So the analysis of one core can assume the other cores are 'busy' but should not depend on the particular execution

sequences (memory accesses) of the software executing on other cores.

One interesting idea here would be to introduce random fluctuations in clock speed, for each individual core. This effectively breaks inter-core pathological cases by breaking the notion of shared time which some pathological cases rely on (ie. a pathological case which requires precise ordering of instructions on a core relative to what is happening on another core; if the notion of time fluctuates between cores, then this probability of maintaining this precise ordering over any period of time effectively drops to zero).

The simple analysis above has focused on synchronous hardware where the clocked instruction represents the 'component' from which task execution times (at a much higher level of time granularity) are derived. Asynchronous (clockless) hardware would drop the level of analysis to an even lower level. Such hardware has already shown that clock-free basic executions can give rise to timely executions at higher levels of abstraction [11], [6]. Counter-intuitively, for real-time applications, it may be that asynchronous hardware may be more predictable than synchronous!

IV. Conclusion

The time scale differences between the basic components of modern hardware and the time bands in which the applications resides means that methods based on the worst-case of the worst-case of worst-case (etc. for each architectural level) cannot deliver temporal predictions that are anything other than hopelessly pessimistic. We must develop methods from which worst-case behaviour can emerge from the average behaviour of lower level components. This short paper has illustrated the potential benefits thats can accrue from randomising the behaviour of the hardware components. Whether such components will emerge remains an open question.

Acknowledgements

This work is funded in part by the UK EPSRC via the Tempo Project. The authors would like to thanks Bev Littlewood and Paul Cairns for advice on the Central Limit Theorem.

References

[1] G. Bernat, M. Newby, and A. Burns. Probabilistic timing analysis, an approach using Copulas. *Journal of Embedded Computing*, 1(2):179–194, 2005.

[2] A. Burns and I.J. Hayes. A timeband framework for modelling real-time systems. *Real-Time Systems Journal*, 45(1–2):106–142, June 2010.

[3] A. Burns and B. Littlewood. Reasoning about the reliability of multi-version, diverse real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 73–81, 2010.

[4] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Proceedings 22nd IEEE Real-Time Systems Symposium*, 2001.

[5] E.Quiones, E. Berger, G. Bernat, and F. Cazorla. Using randomised caches in real-time systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 129–138, 2009.

[6] M. Ferringer. Towards self-timed logic in the time-triggered protocol. In *International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 136–141, 2010.

[7] D Griffin and A Burns. Realism in statistical analysis of worst case execution times. In *10th Intl. Workshop on Worst-Case Execution Time Analysis*, pages 49–57, July 2010.

[8] E. J. Gumbel. *Statistics of Extremes*. Columbia University Press, 1958.

[9] E.A. Lee. Compositional timing in concurrent, parallel, and distributed real-time systems. In *Keynote Talk, CRTS 2010*, 2010.

[10] R.B. Nelsen. *An introduction to Copulas*. Springer, 1998.

[11] J. Spars and S. Furber. *Principles of Asynchronous Circuit Design - a Systems Perspective*. Kluwer Academic, 2001.

[12] S. Stepney, F. Polack, and H. Turner. Engineering emergence. In *ICECCS 2006: 11th IEEE International Conference on Engineering of Complex Computer Systems, Stanford, CA, USA, August 2006*, pages 89–97. IEEE, 2006.