Reality Check: the Need for Benchmarking in RTS and CPS

Marco Di Natale
Scuola Superiore S. Anna
email: marco@sssup.it

Chuansheng Dong
McGill University
email: chuansheng.dong@mail.mcgill.ca

Haibo Zeng
McGill University
email: haibo.zeng@mcgill.ca

Compared with other embedded systems research communities, including electronic design automation (EDA), compilers, or signal processing, the real-time systems (RTS) and the more recent cyber-physical systems (CPS) communities suffer from the lack of uniformly accepted benchmarks and even models on which to evaluate algorithms and solutions. Correspondingly, and not surprisingly, there is lack of consensus on what is *realistic* or *applicable*.

The problem is not minor as most researchers in the area continue to experience such a difficulty. It is also quite fundamental, touching upon very well established results, models, and techniques. We shortly review (and possibly challenge) some methods, models, and tools, and then move to a proposal for a shared (and public) repository of test cases and benchmarks.

Of course it is not by chance that our community lacks consensus on benchmarks or realistic system models. The main reason is probably the need for high-level system models, which are often application-specific (as opposed to general-purpose and well established algorithms that make up most of the benchmark libraries for other communities). Any application-specific model is often considered as part of the company IP (if industrial) and name masking is typically not enough to remove concerns. Hence, the only option is to patiently wait until the product (line) gets out of the market. A secondary reason is that citation-oriented metrics are inevitably leading to diminishing returns from realistic system-level (applicable) research and possibly to more limited interaction(s) with the industrial world.

Even if available, an industrial task model is probably not the best benchmark. Industrial task models are often the product of a designer's interpretation of the code-level solution to a functional problem. In many cases, the designer will seek a simplified task model to reduce concurrency and simplify its job (dealing with race conditions). The task model should be considered as the product of a design synthesis or optimization activity and the functional model, when existing, should always be the preferrable input.

I. System (task and message) models

Several methods and algorithms that are developed in literature are often validated against randomly generated task sets, where the task set complies with a defined activation, synchronization (dependency) and communication model. Quite often task sets are treated as abstract entities and uniform coverage of the space of attributes is sought. For example, confuting the analysis in [6] that stated that the average least upper bound for fixed-priority rate monotonic scheduling is 89%, the UUniFast algorithm [1] was developed to randomly generate task sets with a more accurately *uniform* distribution of task periods and utilizations. In reality, task periods are not arbitrarily selected, but are often the result of oversampling and undersampling at fixed ratios, giving rise to (pseudo-)harmonic sets or constranied by the processing rates of commercial off-the shelf smart sensors.

Also, the community has analyzed several task models as logical evolution of existing ones or as a possible representation of program-level constructs. Several of these did not originate from an application problem defined as a set of tasks but as a possible variation/extension of previous models. The models took a life of their own and researchers are often dealing with the backward work of finding a real-world motivation or match to the task model (and scheduling problem) they know how to solve.

In many real-world cases, the communication and synchronization problems are either much simpler (simple communication by sampling), or not easily represented by using the existing models (such as the activation modes in the Controller Area Network interaction layer, and the scheduling problems defined by synchronous state machine models [8]). As a further example, the model of linear transactions (possibly with offsets) is much more studied but much less common than a more general model of tasks and messages interacting in a graph pattern.

A. Realism and Details

In general, any engineering model should be detailed enough to capture the system attributes of interest to provide an accurate analysis of selected properties. Although aspects like the impact of cache, context switch, interrupt handling delays, and other architectural aspects can play a significant roles in some cases, they are often hidden by considering them (usually after upper-bounding) as part of other system parameters (such as the WCET of tasks) or neglected altogether, under the assumption that their impact is negligible.

Common examples include the assumption that context switch costs are much smaller than the execution times of tasks. However, measures of typical context switch times

for real operating systems on real execution platforms are not often readily available to validate this assumption and experimental results on real OS are required.

Similarly, tradeoffs between policies should be defined by the actual times that are used to perform operations. One example is the comparison between locked-based multiprocessor synchronization protocols like MPCP and MSRP. Unfortunately, a large body of study on this topic has assumed that the critical sections can be larger than $500\mu s$ or even $1ms$, which is probably not realistic for critical sections implementing access to shared memory buffers. Even with a very large data buffer (around 1k byte), the access time is measured to be in the range of $50\mu s$ on a 120MHz embedded microprocessor. Examples of (possibly longer) critical sections in (real-world) applications are required.

B. Random Task/System Generation

When a randomly generated system configuration is acceptable, standarized tools are needed to make experiments repeatable. Among the available generators, TGFF (Task Graph For Free) [2] and SMFF (System Model For Free) [7] are the most popular.

TGFF [2] is widely used in research domains other than real-time systems. It generates random task models, including the task parameters (periods and WCETs), the communication topology, and application-level timing constraints (end-to-end deadlines). However, TGFF does not represent the execution platform and the mapping of tasks. To partially address this issue, SMFF [7] covers the description of the entire system, including the hardware architecture, the software applications, and their mapping onto the platform with the associated scheduling and timing parameters.

Of course, the relevance to reality of these random task/system generation tools is largely decided by the parameter settings. It is also restricted by the underlying assumption in the random generation algorithm. For example, TGFF assumes that the task graph is directed acyclic (which is not suitable for tasks captured by cyclic graphs such as finite state machines), while SMFF assigns task activation periods assuming a uniform distribution between the specified minimum and maximum values.

C. Available Benchmarks

EEMBC [3] is one of the earliest efforts towards developing performance benchmarks for use in embedded systems. It organizes the test suites by specific focus of embedded systems hardware and software development, for application domains including automotive, consumer electronics, and telecommunications, or to address specific design concerns such as energy and floating point performance.

MiBench [4], collected in the early 2000s, follows the model of EEMBC by dividing a set of 35 applications (available as C code) into 6 categories, including automotive and industrial control, consumer devices, office automation, networking, security, and telecommunications.

For a specific purpose of evaluating WCET analysis methods and tools, the Mälardalen WCET research group maintains a benchmark [5], containing 35 programs (provided in C source files) collected from several different research groups and tool vendors around the world.

However, these benchmarks have their limitations. EEMBC is not freely available, access requires a membership with the associated cost. MiBench (besides its out-of-date) and Mälardalen benchmark are collections of programs rather than entire systems with a defined task structure.

II. A website for benchmark

Of course, this lack of benchmarks asks for concrete action. We present our project for the construction of a website [9] meant to store, classify, manage, and provide access to tools, tests and examples constructed and accessible using an open text-based format.

The website (or portal if preferred) is organized with sections for tools (for random generation) and real-world examples or case studies, classified in turn along two dimensions: type of models (Functional, describing functions and signals; Task, with tasks and messages; and Platform models, with physical architectures and possibly OS, device driver or communication stack models), and according to the execution platform (single-core, multicore, or distributed).

Examples are available and can be provided in an open text format, according to the description provided in the format section. We started collecting message sets and application descriptions from automotive systems. The website will be hosted at McGill University [9], with the collection of the above mentioned tools and benchmarks. Please feel invited to contribute with representative test cases and links.

References

[1] E. Bini and G. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Sys.*, 30(1-2):129–154, May 2005.

[2] R. Dick, D. Rhodes, and W. Wolf. TGFF: task graphs for free. Available at http://ziyang.eecs.umich.edu/ dickrp/tgff/

[3] The Embedded Microprocessor Benchmark Consortium. Website: http://www.eembc.org/

[4] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. Available at http://www.eecs.umich.edu/mibench/

[5] WCET benchmarks. Available at http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[6] J.P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. 10th Real-Time Systems Symposium*, 1989.

[7] M. Neukirchner, S. Stein, and R. Ernst. SMFF: System Models for Free. Available at http://smff.sourceforge.net/

[8] H. Zeng and M. Di Natale. Schedulability Analysis of Periodic Tasks Implementing Synchronous Finite State Machines. *Proc. 23rd Euromicro Conference on Real-Time Systems*, 2012.

[9] An Open Repository for Real-Time Benchmarks. Available at http://www.ece.mcgill.ca/hzeng3/benchmarks.html