2

Performance Metrics for Real-time Systems

So-called benchmark programs or program sets [62, 70, 75] have been selected or designed for the purpose of testing and/or measuring the performance of computing systems in one way or another. They should assist in design or purchasing decisions and in the testing or acceptance stages. Benchmarks are, however, infamous for being both controversial and subjective.

Different kinds of real-time benchmarks have been developed for different areas and types of use:

High-performance computing systems (e.g. signal, voice and image processing, artificial intelligence or numerical analysis)
High-availability computing systems (e.g. control systems, avionics, database or telecommunication systems)

The magnitude of time granularity varies depending on the field of use and type of application. Also, structural awareness of the benchmark application is more important for high-availability computing systems and is present in their benchmarks.

In the sequel, first some background information on performance measurement is given, followed by a selection of some standard benchmarks, which have been devised to evaluate real-time systems. They shall be classified as benchmark programs, timing analysers and performance monitors.

2.1 Benchmarks and RTOS Standardisation

Devising benchmark programs for real-time systems is difficult due to the late standardisation of real-time operating systems. Many different real-time operating systems are currently being used, each one supporting its own specific system calls. The portability of real-time benchmark programs is, therefore, very limited. Some years ago, the set of POSIX 1003.4 standards [7] was defined for real-time operating systems. This should assist in building real-time operating systems with standard interfaces and standard benchmark applications.

To give an example, in [53] a set of benchmark criteria relevant for POSIX systems was defined. These criteria are subdivided into two categories:

1. Those that measure the determinism of the operating system (called "deterministic benchmarks")
2. Those that measure the latency of particular important operations (called "latency benchmarks")

These benchmarks test core operating system capabilities and are independent of any actual application. "Deterministic benchmarks" measure the response time of an operating system, determine if a thread can respond in a deterministic fashion and measure the maximum kernel blocking time. 'Latency benchmarks", on the other hand, are designed to measure the context-switching time between threads and processes, the possible throughput of data between processes and threads and the latency of POSIX real-time signals.

2.2 DIN 19242 Performance Measurement Methods

The German Standardisation Institute's DIN 19242 standard (Part 1) [12] defines performance, measurable parameters and measuring methods for process control computers. Based on these, in [73, 4] a test method and 14 programming examples in C, FORTRAN, Pascal and PEARL are given to measure the parameters defined in DIN 19242. The test suite contains program segments for the following functions: multiplication, conditional branch, sine calculation, matrix inversion, bit pattern detection, interrupt handling with program start, I/O operation on peripheral storage, digital I/O, dialogue response at a terminal, message-oriented synchronisation between two tasks, record-oriented reading from and writing to a (random access) file, data transmission with an error-free transmission circuit and generation of executable code with a subsequent program start. The suite was employed in [8] to compare the performance of a personal computer with that of a microVAX II based on representative performance criteria and appropriate loads. Later, the suite was included in the successor parts of the mentioned standard where, based on applications, appropriate test programs were grouped. In DIN 19243 [13] standard finally measurement and control methods, basic functions of automation with process control computers, appropriate analogue and binary quantities, as well as data acquisition, processing and output were defined.

For a broader application area the DIN 66273 [16] standard was written, which defines data processing tasks, measurement methods, and rating of data processing performance of computing systems, whereby it concentrates on sensible and measurable performance quantities (e.g. response times).

The description of the method to measure execution time according to the DIN 19242 standard is given here as an example. It is based on the following presumptions (see Fig. 2.1):

- Any process control computer runs a multitasking operating system
- There are two programs competing for the CPU — a low-priority process executing a CPU-intensive task that does not require additional resources, and a high-priority process executing the measured task hereby blocking the execution of the former
- The low-priority task's execution time (t_{LP}) is delayed for a time interval being as long as the execution time of the (high-priority) measured process (t_{HP})

The measurement method is the following. First, only the execution time of the low-priority process is measured, then the execution time of both processes is measured (T). The difference of the measured execution times ($T_{meas.}$) represents the execution time of the high-priority process, which also includes the necessary context switches.

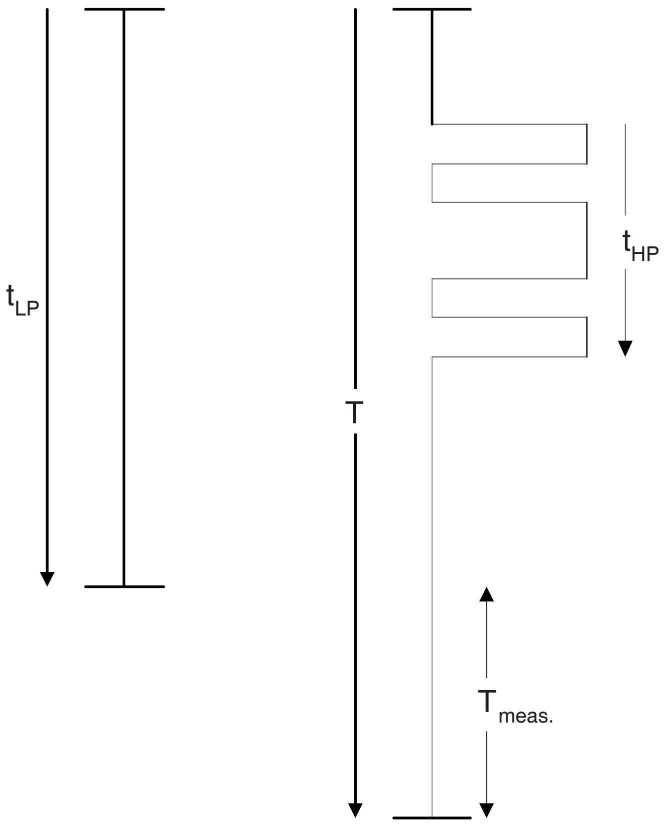

Fig. 2.1. Execution time measurement based on DIN 19242

2.3 Benchmark Programs

2.3.1 Rhealstone Benchmark

The Rhealstone benchmark [42, 43, 44] is a set of six C programs, with which the following characteristics of real-time operating systems can be measured:

Average task switch time: average time for switching between two independent active processes (tasks) of the same priority (i.e. effectiveness of (re-) storing process contexts)
Average pre-emption time: average time for switching to a previously inactive higher-priority process (i.e. effectiveness of schedule-dispatch algorithm)
Average interrupt latency: average time between the occurrence of an interrupt at the CPU and the execution of the corresponding interrupt service routine's first instruction (i.e. effectiveness of dynamic priority assignment and scheduling)
Semaphore shuffle time: average time between a request to lock a semaphore, which is locked by another process, and the time when its request is granted
Deadlock break time: average time to resolve a deadlock that occurs when a lower-priority process holding a resource is pre-empted by a higher-priority process also requiring the same resource
Inter-task message latency: average delay that occurs when sending a message between two processes

The measured times are joined in a common mean value $t^{'} = \frac{1}{6}(t_1 + t_2 + t_3 + t_4 + t_5 + t_6)$ expressed in seconds. The so-called Rhealstone value is defined as the inverse of this value: $\frac{1}{t^{'}}$. Hence, the comparative performance of a real-time system is greater if its Rhealstone value is greater.

The major drawbacks of the Rhealstone benchmark are: first, it measures average times instead of the more important worst-case values and, secondly, the Rhealstone benchmark states that these acquired times should be combined into a single figure by the weighted summation of the measured values. No details are given as to determine the weight factors. More criticism was pronounced by Kasten [44].

2.3.2 MiBench

MiBench [23] is a set of 35 applications typical for embedded systems and divided into six suites, each one targeting a specific area of the market for embedded systems: (1) automotive and industrial control, (2) consumer devices, (3) office automation, (4) networking, (5) security and (6) telecommunications. All programs are available in standard C source code, and portable to any platform that has compiler support. Where appropriate, MiBench provides small and large data sets. The small data set represents a lightweight application of the benchmark, whereas the large data set provides a real-world application. While sharing some common properties with the EEMBC [18] suite, MiBench comes with freely available source code. For hard real-time systems, only the first suite is useful.

2.4 Timing Analysers

To determine timeliness, timing analyses are carried out either off-line or on-line. While off-line timing analysis mainly concentrates on profiling compiled code to discover bottlenecks and to determine a priori timeliness, for on-line timing analyses execution times of the actual application programs or of sets of benchmark programs with known properties are measured.

Owing to the complexity of contemporary real-time applications and processors used, it is difficult to determine exact program execution times off-line — even on the assembly language level. It is necessary to limit the use of high-level language constructs considerably, and to preferably use simpler microprocessors without caches or pipelines in order to be able to obtain useful results. Otherwise, results will be either too pessimistic, or applicable in special cases or for smaller subsystems, only. Hence, off-line analysis has been found to be of limited use for contemporary real-time applications and, although not as precise, overall on-line timing analysis is predominantly employed.

2.4.1 Hartstone Benchmark

The Hartstone benchmark [1] is a set of operational requirements for a synthetic application used to test hard real-time systems. It was written in Ada at Carnegie Mellon University.

The Hartstone benchmark requirements define several test series of processes, viz., periodic with hard deadlines, aperiodic with soft or hard deadlines and synchronisation processes without deadlines. Critical for the Hartstone benchmark are the execution times of synchronisation processes, servicing periodic and aperiodic client processes, which are the sums of the execution times of the critical sections and the independent execution times of the server processes. Each test in a series either meets its deadline, or misses one or more deadlines, and thus fails. There are five test series:

1. Periodic tasks, harmonic frequencies
2. Periodic tasks, non-harmonic frequencies
3. Periodic tasks, harmonic frequencies with aperiodic processing
4. Periodic tasks, harmonic frequencies with synchronisation
5. Periodic tasks, harmonic frequencies with aperiodic processing and synchronisation

For each test series there exists a "baseline task set" which exactly defines the computational load, timing requirements, and period lengths. A test begins with its baseline set, and must fulfil the specified timing requirements. In any subsequent test a single parameter of the tasks in the set is changed. A test is repeated until a "stop condition", usually associated with a deadline or response time miss, is reached. The results of a test series are the task set parameter settings at the time when a stop condition is reached with the highest computational load. As a synthetic computational load the Whetstone benchmark [11] may be used.

Carefully interpreting the results obtained with this test series, one should be able to consider the suitability of a complete platform (hardware and system software) for real-time applications [77]. While the described Hartstone benchmark was devised for single-processor systems, it was later enhanced for distributed environments [41] taking also processor assignment strategies and interprocessor communication into consideration.

2.4.2 SNU Real-time Benchmarks

One of the more recent benchmarks was developed at Seoul National University [68]. It has been devised to run without an operating system. All its functions are provided in the form of C source code. The benchmark programs fulfil the following structural constraints: no unconditional jumps, no exit from loop bodies, no switch statements, no while constructs, no multiple expressions joined by conjunction or disjunction operators and no library calls. As a compilation of some numerical calculation programs (binary search, Fibonacci series, insertion sort, etc.) and digital signal processing algorithms (CRC, FFT, JPEG, etc.) the benchmark is to be used for worst-case timing analysis. Its benefit is independence of target platforms, and its drawback is the limited usefulness in more complex environments with operating systems. From the foreseen experimental set-up (see Fig. 2.2) it was also possible to classify the benchmark programs as the software monitors to be discussed in the next section. The "modified LCC" and "timing analyser" are the core components.

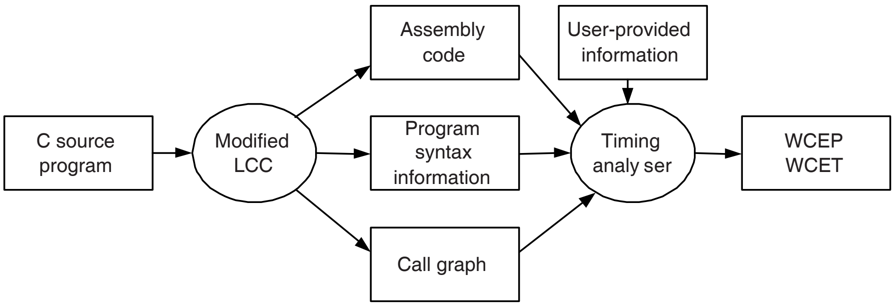

Fig. 2.2. Experimental environment of SNU benchmark for worst-case execution time analysis

2.5 Performance Monitors

Some reasons for monitoring data on the performance of a real-time system are to determine workload characteristics, to verify correctness and to determine reliability. Performance monitoring consists of gathering data on a system during its operation and condensing them into a form suitable for interpretation. Monitoring tools fall into one of the four categories: hardware monitors, software monitors, firmware monitors and hybrid monitors [48]. One may consider most of the mentioned benchmarks as software monitors. In order to obtain more precise results, one may introduce external monitoring devices (hardware monitors), or observe timing events at a lower level (firmware monitors). When hardware monitors have some internal logic for the interpretation of measured times we speak about hybrid monitors.

2.5.1 PapaBench

The PapaBench suite [52] is a real-time benchmark freely available under the GNU General Public License. It is based on the *Paparazzi* project that represents a real-time application developed for different Unmanned Aerial Vehicles (UAV). The benchmark was designed to estimate Worst-Case Execution Times (WCET) and to analyse schedulability. Its programs are simple in structure. With the exception of two while loops, they are composed of not nested for loops with fixed upper bounds, only. Thus, WCET analysis is simplified, and WCET estimation can come closer to the real WCET.

2.5.2 RT_STAP Benchmark

The "Real-Time Space-Time Adaptive Processing" (RT_STAP) benchmark [10] is used to evaluate the application of scalable high-performance computers to real-time implementations of space-time adaptive processing (STAP) on embedded platforms. The STAP techniques are used to support clutter and interference cancelation in airborne radars. RT_STAP is an example of a compact application benchmark that employs a real-time design-to-specification methodology. The scalability study outlined in this benchmark varies the sophistication and computational complexity of the adaptive algorithms to be implemented. RT_STAP provides hard, medium and easy benchmark cases based on three post-Doppler adaptive processing algorithms: higher-order Doppler-factored STAP, first-order Doppler-factored STAP and post-Doppler adaptive Displaced Phase Centre Antenna.

2.5.3 Benchmarks Oriented at Databases and Telecommunication

As compared to traditional benchmarks for On-Line Analytical Processing (OLAP) applications, originating from banking and business applications (e.g. TPC-B, TPC-C), telecommunication applications (e.g. GSM, ATM communication networks) deal with smaller amounts of data. Since these data mostly fit nicely into the main memories of computers, the mentioned traditional benchmarks are not suitable, as they focus mainly on the performance and efficiency of disk access (e.g. TPC-C) and do not consider service times as correctness criteria.

Hence, for telecommunication applications new benchmarks (e.g. [49]) have been developed based on the mentioned ones (e.g. TPC-B) and on predispositions. Here, a similar approach as for database applications is chosen. Test transactions are defined, which are performed in a database of some service provider(s). The Atomicity, Consistency, Isolation and Durability (ACID) properties of transaction processing must be assured by a system under test while the benchmark is being run. The metric's value is reported in transactions per second (*tpsT*), i.e. the total number of successfully committed transactions divided by the elapsed time of the measurement interval. A committed transaction is only considered successful if it has both started and completed within the measurement interval and before its deadline. The transaction success ratio (*successT*) is the total number of successfully committed transactions divided by the total number of transactions entered to a system under test during the measurement interval. The transaction miss ratio (*missT*) then equals $1 - successT$. It represents all unsuccessfully ended transactions.

2.6 Concluding Remarks on Test Methods and Benchmarks

The general division of benchmarks for the two main streams of real-time applications holds for most cases. We may say that application-oriented high-performance benchmarks generally offer more realistic results, since real applications do not differ much from the way the benchmarks use the basic operations. Structurally oriented high-availability benchmarks, on the other hand, also speculate on different scenarios within the applications in which they use the basic operations of the systems under test. Since two applications are never the same, the results obtained adhere more to the properties of the systems than the applications. However, one may also speculate on the relevant results based on the knowledge of a benchmark's structure and the known structure of an application, or even integrate the application into the benchmark. This would, of course, produce entirely accurate results, but is not always possible.

Assessing the capabilities of the above mentioned benchmarks and monitoring methods yields the conclusion that all of them are unable to render the results really needed, because the speed with which a certain arithmetic operation is performed is as insignificant for the evaluation of a real-time application as a whole as, say, an average task switching time or data obtained from test runs, because the latter usually cannot cover all worst-case conditions. Therefore, in the next chapter we take a more thorough look at the fundamental principles of real-time computing to determine performance criteria that really matter.

Although benchmarks have a somewhat bad reputation, it would be good for the real-time community if a group were formed, comprising both system vendors and users, to specify, promote and report benchmarks. System vendors should feel motivated to participate, since this would provide them with an opportunity to spotlight their systems. Users already investing resources in benchmarking systems would feel more motivated by investing in an industry effort that would enable them to influence and leverage the results, ultimately reducing the overall efforts required to accurately assess the desired properties of their computing systems (components). This approach was already successful with the DIN 66273 standard.