microbenchmark: A package to accurately benchmark *R* expressions

Olaf Mersmann[1*], **Sebastian Krey**[1]

1. TU Dortmund, Department of Statistics
*Contact author: olafm@datensplitter.net

Keywords: Benchmarking, Timing, High Performance Computing

We present the *R* package **microbenchmark**. It provides functions to accurately measure the execution time of *R* expressions. This enables the user to benchmark and compare different implementation strategies for performance critical functions, whose execution time may be small, but which will be executed many times in his program. In contrast to the often used `system.time` and `replicate` combination, our package offers the following advantages: firstly it attempts to use the most accurate method of measuring time made available by the underlying operating system, secondly it estimates to overhead of the timing routines and subtracts it from all measurements automatically and lastly it provides utility functions to quickly compare the timing results for different expressions.

In our presentation we will first describe the implementation of the timing routines for each platform (Windows, MacOS X, Linux and Solaris) and highlight some of the difficulties and limitations when measuring sub-millisecond execution times. Topics covered include strategies for dealing with CPU frequency scaling and multi-core system, why elapsed time is measured and not CPU time, why time is measured and not clock cycles as well as ways to estimate the granularity of the underlying timing routine. We conclude with a practical guideline for benchmark experiments on each of the operating systems mentioned.

After the viability of timing very short running *R* expressions has been established, some examples are provided to show why it is useful to study the performance characteristics of such expressions at all. First off, a baseline for the "speed" of the *R* interpreter is established by measuring the time it takes to execute the simplest possible function[1]. We consider this to be the equivalent of the cycle time of a microprocessor. Next we extend this idea to S3 and S4 methods to see how much overhead one incurs for the method dispatch in comparison to a simple function call. Afterwards we investigate the time it takes to generate a vector containing the numbers from 1 to n (for different n) using the different methods available in base *R*. Time permitting we conclude with a real world example where using high-level vectorized *R* functions actually hurt performance when compared to a naive implementation using low level operations.

[1] The simplest possible function being `function() NULL`.