

Package ‘microbenchmark’

September 4, 2024

Title Accurate Timing Functions

Description Provides infrastructure to accurately measure and compare the execution time of R expressions.

URL <https://github.com/joshuaulrich/microbenchmark/>

BugReports <https://github.com/joshuaulrich/microbenchmark/issues/>

License BSD_2_clause + file LICENSE

Depends R (>= 3.2.0)

Imports graphics, stats

Suggests ggplot2, multcomp, RUnit

SystemRequirements On a Unix-alike, one of the C functions `mach_absolute_time` (macOS), `clock_gettime` or `gethrtime`. If none of these is found, the obsolescent POSIX function `gettimeofday` will be tried.

ByteCompile yes

Version 1.5.0

NeedsCompilation yes

Author Olaf Mersmann [aut],
Claudia Beleites [ctb],
Rainer Hurling [ctb],
Ari Friedman [ctb],
Joshua M. Ulrich [cre]

Maintainer Joshua M. Ulrich <josh.m.ulrich@gmail.com>

Repository CRAN

Date/Publication 2024-09-04 18:30:02 UTC

Contents

<code>autoplot.microbenchmark</code>	2
<code>boxplot.microbenchmark</code>	3
<code>get_nanotime</code>	4

microbenchmark	4
microtiming_precision	7
print.microbenchmark	8
summary.microbenchmark	9

Index	10
--------------	-----------

autoplot.microbenchmark

Autoplot method for microbenchmark objects: Prettier graphs for microbenchmark using ggplot2

Description

Uses ggplot2 to produce a more legible graph of microbenchmark timings.

Usage

```
autoplot.microbenchmark(
  object,
  ...,
  order = NULL,
  log = TRUE,
  unit = NULL,
  y_max = NULL
)
```

Arguments

object	A microbenchmark object.
...	Ignored.
order	Names of output column(s) to order the results.
log	If TRUE the time axis will be on log scale.
unit	The unit to use for graph labels.
y_max	The upper limit of the y axis, in the unit automatically chosen for the time axis (defaults to the maximum value).

Value

A ggplot2 object.

Author(s)

Ari Friedman, Olaf Mersmann

Examples

```

if (requireNamespace("ggplot2", quietly = TRUE)) {
  tm <- microbenchmark(rchisq(100, 0),
                      rchisq(100, 1),
                      rchisq(100, 2),
                      rchisq(100, 3),
                      rchisq(100, 5), times=1000L)

  ggplot2::autoplot(tm)

  # add a custom title
  ggplot2::autoplot(tm) + ggplot2::ggtitle("my timings")
}

```

```

boxplot.microbenchmark

```

Boxplot of microbenchmark timings.

Description

Boxplot of microbenchmark timings.

Usage

```

## S3 method for class 'microbenchmark'
boxplot(
  x,
  unit = "t",
  log = TRUE,
  xlab,
  ylab,
  horizontal = FALSE,
  main = "microbenchmark timings",
  ...
)

```

Arguments

x	A microbenchmark object.
unit	Unit in which the results be plotted.
log	Should times be plotted on log scale?
xlab	X axis label.
ylab	Y axis label.
horizontal	Switch X and Y axes.
main	Plot title.
...	Passed on to boxplot.formula.

Author(s)

Olaf Mersmann

get_nanotime	<i>Return the current value of the platform timer.</i>
--------------	--

Description

The current value of the most accurate timer of the platform is returned. This can be used as a time stamp for logging or similar purposes. Please note that there is no common reference, that is, the timer value cannot be converted to a date and time value.

Usage

```
get_nanotime()
```

Author(s)

Olaf Mersmann

microbenchmark	<i>Sub-millisecond accurate timing of expression evaluation.</i>
----------------	--

Description

microbenchmark serves as a more accurate replacement of the often seen `system.time(replicate(1000, expr))` expression. It tries hard to accurately measure only the time it takes to evaluate `expr`. To achieve this, the sub-millisecond (supposedly nanosecond) accurate timing functions most modern operating systems provide are used. Additionally all evaluations of the expressions are done in C code to minimize any overhead.

Usage

```
microbenchmark(  
  ...,  
  list = NULL,  
  times = 100L,  
  unit = NULL,  
  check = NULL,  
  control = list(),  
  setup = NULL  
)
```

Arguments

...	Expressions to benchmark.
list	List of unevaluated expressions to benchmark.
times	Number of times to evaluate each expression.
unit	Default unit used in summary and print.
check	A function to check if the expressions are equal. By default NULL which omits the check. In addition to a function, a string can be supplied. The string 'equal' will compare all values using <code>all.equal</code> , 'equivalent' will compare all values using <code>all.equal</code> and <code>check.attributes = FALSE</code> , and 'identical' will compare all values using <code>identical</code> .
control	List of control arguments. See Details.
setup	An unevaluated expression to be run (untimed) before each benchmark expression.

Details

This function is only meant for micro-benchmarking small pieces of source code and to compare their relative performance characteristics. You should generally avoid benchmarking larger chunks of your code using this function. Instead, try using the R profiler to detect hot spots and consider rewriting them in C/C++ or FORTRAN.

The control list can contain the following entries:

order the order in which the expressions are evaluated. "random" (the default) randomizes the execution order, "inorder" executes each expression in order and "block" executes all repetitions of each expression as one block.

warmup the number of iterations to run the timing code before evaluating the expressions in ... These warm-up iterations are used to estimate the timing overhead as well as spinning up the processor from any sleep or idle states it might be in. The default value is 2.

Value

Object of class 'microbenchmark', a data frame with columns `expr` and `time`. `expr` contains the de-parsed expression as passed to `microbenchmark` or the name of the argument if the expression was passed as a named argument. `time` is the measured execution time of the expression in nanoseconds. The order of the observations in the data frame is the order in which they were executed.

Note

Depending on the underlying operating system, different methods are used for timing. On Windows the `QueryPerformanceCounter` interface is used to measure the time passed. For Linux the `clock_gettime` API is used and on Solaris the `gethrtime` function. Finally on MacOS X the, undocumented, `mach_absolute_time` function is used to avoid a dependency on the CoreServices Framework.

Before evaluating each expression `times` times, the overhead of calling the timing functions and the C function call overhead are estimated. This estimated overhead is subtracted from each measured evaluation time. Should the resulting timing be negative, a warning is thrown and the respective

value is replaced by 0. If the timing is zero, a warning is raised. Should all evaluations result in one of the two error conditions described above, an error is raised.

One platform on which the clock resolution is known to be too low to measure short runtimes with the required precision is Oracle® Solaris on some SPARC® hardware. Reports of other platforms with similar problems are welcome. Please contact the package maintainer.

Author(s)

Olaf Mersmann

See Also

[print.microbenchmark](#) to display and [boxplot.microbenchmark](#) or [autoplot.microbenchmark](#) to plot the results.

Examples

```
## Measure the time it takes to dispatch a simple function call
## compared to simply evaluating the constant \code{NULL}
f <- function() NULL
res <- microbenchmark(NULL, f(), times=1000L)

## Print results:
print(res)

## Plot results:
boxplot(res)

## Pretty plot:
if (requireNamespace("ggplot2")) {
  ggplot2::autoplot(res)
}

## Example check usage
my_check <- function(values) {
  all(sapply(values[-1], function(x) identical(values[[1]], x)))
}

f <- function(a, b)
  2 + 2

a <- 2
## Check passes
microbenchmark(2 + 2, 2 + a, f(2, a), f(2, 2), check=my_check)
## Not run:
a <- 3
## Check fails
microbenchmark(2 + 2, 2 + a, f(2, a), f(2, 2), check=my_check)

## End(Not run)
## Example setup usage
set.seed(21)
```

```
x <- rnorm(10)
microbenchmark(x, rnorm(10), check=my_check, setup=set.seed(21))
## Will fail without setup
## Not run:
microbenchmark(x, rnorm(10), check=my_check)

## End(Not run)
## using check
a <- 2
microbenchmark(2 + 2, 2 + a, sum(2, a), sum(2, 2), check='identical')
microbenchmark(2 + 2, 2 + a, sum(2, a), sum(2, 2), check='equal')
attr(a, 'abc') <- 123
microbenchmark(2 + 2, 2 + a, sum(2, a), sum(2, 2), check='equivalent')
## check='equal' will fail due to difference in attribute
## Not run:
microbenchmark(2 + 2, 2 + a, sum(2, a), sum(2, 2), check='equal')

## End(Not run)
```

microtiming_precision *Estimate precision of timing routines.*

Description

This function is currently experimental. Its main use is to judge the quality of the underlying timer implementation of the operating system. The function measures the overhead of timing a C function call rounds times and returns all non-zero timings observed. This can be used to judge the granularity and resolution of the timing subsystem.

Usage

```
microtiming_precision(rounds = 100L, warmup = 2^18)
```

Arguments

rounds	Number of measurements used to estimate the precision.
warmup	Number of iterations used to warm up the CPU.

Value

A vector of observed non-zero timings.

Author(s)

Olaf Mersmann

```
print.microbenchmark Print microbenchmark timings.
```

Description

Print microbenchmark timings.

Usage

```
## S3 method for class 'microbenchmark'
print(x, unit, order, signif, ...)
```

Arguments

x	An object of class microbenchmark.
unit	What unit to print the timings in. Default value taken from to option microbenchmark.unit (see example).
order	If present, order results according to this column of the output.
signif	If present, limit the number of significant digits shown.
...	Passed to print.data.frame

Note

The available units are nanoseconds ("ns"), microseconds ("us"), milliseconds ("ms"), seconds ("s") and evaluations per seconds ("eps") and relative runtime compared to the best median time ("relative").

If the multcomp package is available a statistical ranking is calculated and displayed in compact letter display from in the cld column.

Author(s)

Olaf Mersmann

See Also

[boxplot.microbenchmark](#) and [autoplot.microbenchmark](#) for a plot methods.

Examples

```
a1 <- a2 <- a3 <- a4 <- numeric(0)

res <- microbenchmark(a1 <- c(a1, 1),
                      a2 <- append(a2, 1),
                      a3[length(a3) + 1] <- 1,
                      a4[[length(a4) + 1]] <- 1,
                      times=100L)

print(res)
```



```
## Change default unit to relative runtime
options(microbenchmark.unit="relative")
print(res)
## Change default unit to evaluations per second
options(microbenchmark.unit="eps")
print(res)
```

summary.microbenchmark

Summarize microbenchmark timings.

Description

Summarize microbenchmark timings.

Usage

```
## S3 method for class 'microbenchmark'
summary(object, unit, ..., include_cld = TRUE)
```

Arguments

object	An object of class microbenchmark.
unit	What unit to print the timings in. If none is given, either the unit attribute of object or the option microbenchmark.unit is used and if neither is set "t" is used.
...	Ignored
include_cld	Calculate cld using multcomp::glht() and add it to the output. Set to FALSE if the calculation takes too long.

Value

A data.frame containing the aggregated results.

Note

The available units are nanoseconds ("ns"), microseconds ("us"), milliseconds ("ms"), seconds ("s") and evaluations per seconds ("eps") and relative runtime compared to the best median time ("relative").

See Also

[print.microbenchmark](#)

Index

`all.equal`, 5
`autoplot.microbenchmark`, 2, 6, 8
`boxplot.microbenchmark`, 3, 6, 8
`get_nanotime`, 4
`identical`, 5
`microbenchmark`, 4
`microtiming_precision`, 7
`print.microbenchmark`, 6, 8, 9
`summary.microbenchmark`, 9