A Machine Profile Summary

John C. Nash * Arkajyoti Bhattacharjee †

14/06/2021 revised 18/01/2022

Providing a characterization of a particular computing environment for R calculations

If we want to compare results computed on different machines, we need a way to provide measures and identifiers of the particular computing environment at hand. This write-up attempts to summarize some possibilities available in the R statistical software and language. Note that one of our interests is to have tools that work in R **across different platforms**. That is, we want to be able to call an R function to get machine and computing environment characterizations regardless of the particular platform which we are currently using. Moreover, we want to have a succinct output of such information.

Some possible information desired

There are multiple ways to get information of the type useful for characterizing the computing infrastructure used for a particular test or timing. However, we need to ensure that identifiers of particular pieces of information are unique, and that common identifiers really do specify identical, or at least equivalent, information. Otherwise, new names for some pieces of our set of capability information are needed.

Temporal information

When we wish to analyze performance results, it is extremely helpful to have information on when the calculations were performed. Most computations should have a time/date stamp that is unlikely to be confused with any other, at least in combination with a machine name and/or other tags. Here is one suggestion based on the R `Sys.time()` function.

```
#get a timestamp
tsstr <- format(Sys.time(), "%Y/%m/%d|%H:%M") # tsstr == time stamp string in form YYYYmmddHHMM
cat("Date and Time stamp:", tsstr, "\n")
```

```
## Date and Time stamp: 2022/06/13|18:30
```

Identifier for the computing software environment

Some of the important information elements concerning the computing environment that we need for reporting tests are

- a machine name. While most systems (and Linux in particular) offer to let the user provide a machine name, there are generally defaults that many people accept, and these are often uninformative and may be non-unique. We note that VirtualBox with a Windows 10 guest machine gave the name DESKTOP-HF4CKVA. Where this name was generated we are not sure. Settings / System allows "rename this PC".

*retired professor, Telfer School of Management, University of Ottawa
†Department of Mathematics and Statistics, Indian Institute of Technology, Kanpur

- operating system and version. For Linux systems, we note that besides the distribution variant, e.g., Linux Mint 20.3 MATE 64 bit, it is necessary to add the Linux kernel identifier, e.g., 5.13.0-25-generic, from the `uname` command.
- compiler or interpreter version. For our needs, it is obvious that the R version will be important. However, if any code is compiled or linked to libraries, we would like to know that. In particular, versions of compilers (e.g., gfortran, gcc) or BLAS or LAPACK libraries used will affect performance. Linux generally displays the BLAS and LAPACK **filenames** in `sessionInfo()`, but to get the version information, one needs to dig deeper, for example, using operating system commands. For the present, we will omit such extra information unless we can extract it easily within R.
- specific variations, if any, that should be noted. Here we may want to note if a machine is running background tasks, or if some special steps have been taken to speed up or slow down the operation e.g., overclocking.

Hardware information

Some of the factors relating to hardware that could be important are:

- cpu (i.e., processor)
- number of cores
- operating cycle speed (this is sometimes specific to the cpu specified)
- RAM size total
- RAM size available, possibly indicating swap space and usage
- RAM speed (which could be variable)
- GPU (i.e, additional processors) information.

Software information

Clearly it is important to identify which software packages are active, and the R `sessionInfo()` command is useful for this. However, there may be additional features that should be mentioned if the user has modified the software in any way.

The use of tools to carry out computations in parallel, using multiple cores or GPUs, is clearly an issue. In our opinion, such possibilities are not well supported by common R functions, or indeed in other computing languages, though we would very much like to hear of tools to provide such information.

R tools for machine information

sessionInfo()

```
si <- sessionInfo()
si <- as.vector(si)
si
```

```
## R version 4.1.3 (2022-03-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22000)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_India.1252  LC_CTYPE=English_India.1252
## [3] LC_MONETARY=English_India.1252 LC_NUMERIC=C
## [5] LC_TIME=English_India.1252
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.1.3  magrittr_2.0.1  fastmap_1.1.0   cli_3.2.0
##  [5] tools_4.1.3     htmltools_0.5.2 rstudioapi_0.13 yaml_2.3.5
##  [9] stringi_1.7.6   rmarkdown_2.13  knitr_1.37      stringr_1.4.0
## [13] xfun_0.30       digest_0.6.29   rlang_1.0.2     evaluate_0.15
```

benchmarkme

An important tool for acquiring machine information is the CRAN package `benchmarkme` (Gillespie (2022)). We found a minor bug in the `get_ram()` function of this package for some Windows 10 versions. It has been corrected in the latest version 1.0.8, with contributions from one of the authors (AB).

?? Need to comment on

- uses `proc.time()` for timings
- has single core and parallel timings
- ?? does it do GPUs
- provides far too much for most needs i.e., a succinct summary of how capable a computing system should be

```
library(benchmarkmeData)
library(benchmarkme)
ls(package:benchmarkmeData)
```

```
## Warning in ls(package:benchmarkmeData): 'package:benchmarkmeData' converted to
## character string
```

```
## [1] "get_datatable_past" "is_blas_optimize"   "make_data_set"
## [4] "move_files"         "past_results"       "past_results_v2"
## [7] "plot_past"          "select_results"     "summarise_results"
```

```
ls(package:benchmarkme)
```

```
## Warning in ls(package:benchmarkme): 'package:benchmarkme' converted to character
## string
```

```
##  [1] "benchmark_io"                "benchmark_matrix_cal"
##  [3] "benchmark_matrix_fun"        "benchmark_prog"
##  [5] "benchmark_std"               "bm_matrix_cal_cross_product"
##  [7] "bm_matrix_cal_lm"            "bm_matrix_cal_manip"
##  [9] "bm_matrix_cal_power"         "bm_matrix_cal_sort"
## [11] "bm_matrix_fun_cholesky"      "bm_matrix_fun_determinant"
## [13] "bm_matrix_fun_eigen"         "bm_matrix_fun_fft"
## [15] "bm_matrix_fun_inverse"       "bm_parallel"
## [17] "bm_prog_escoufier"           "bm_prog_fib"
## [19] "bm_prog_gcd"                 "bm_prog_hilbert"
## [21] "bm_prog_toeplitz"            "bm_read"
## [23] "bm_write"                    "create_bundle"
## [25] "get_available_benchmarks"    "get_byte_compiler"
## [27] "get_cpu"                     "get_linear_algebra"
## [29] "get_platform_info"           "get_r_version"
## [31] "get_ram"                     "get_sys_details"
## [33] "is_blas_optimize"            "plot_past"
## [35] "rank_results"                "sample_results"
## [37] "upload_results"
```

```
lsf.str("package:benchmarkme")
```

```
## benchmark_io : function (runs = 3, size = c(5, 50), tmpdir = tempdir(), verbose = TRUE,
##     cores = 0L)
## benchmark_matrix_cal : function (runs = 3, verbose = TRUE, cores = 0L)
## benchmark_matrix_fun : function (runs = 3, verbose = TRUE, cores = 0L)
## benchmark_prog : function (runs = 3, verbose = TRUE, cores = 0L)
## benchmark_std : function (runs = 3, verbose = TRUE, cores = 0L)
## bm_matrix_cal_cross_product : function (runs = 3, verbose = TRUE)
## bm_matrix_cal_lm : function (runs = 3, verbose = TRUE)
## bm_matrix_cal_manip : function (runs = 3, verbose = TRUE)
## bm_matrix_cal_power : function (runs = 3, verbose = TRUE)
## bm_matrix_cal_sort : function (runs = 3, verbose = TRUE)
## bm_matrix_fun_cholesky : function (runs = 3, verbose = TRUE)
## bm_matrix_fun_determinant : function (runs = 3, verbose = TRUE)
## bm_matrix_fun_eigen : function (runs = 3, verbose = TRUE)
## bm_matrix_fun_fft : function (runs = 3, verbose = TRUE)
## bm_matrix_fun_inverse : function (runs = 3, verbose = TRUE)
## bm_parallel : function (bm, runs, verbose, cores, ...)
## bm_prog_escoufier : function (runs = 3, verbose = TRUE)
## bm_prog_fib : function (runs = 3, verbose = TRUE)
## bm_prog_gcd : function (runs = 3, verbose = TRUE)
## bm_prog_hilbert : function (runs = 3, verbose = TRUE)
## bm_prog_toeplitz : function (runs = 3, verbose = TRUE)
## bm_read : function (runs = 3, size = c(5, 50), tmpdir = tempdir(), verbose = TRUE)
## bm_write : function (runs = 3, size = c(5, 50), tmpdir = tempdir(), verbose = TRUE)
## create_bundle : function (results, filename = NULL, args = NULL, id_prefix = "")
## get_available_benchmarks : function ()
## get_byte_compiler : function ()
## get_cpu : function ()
## get_linear_algebra : function ()
## get_platform_info : function ()
## get_r_version : function ()
## get_ram : function ()
## get_sys_details : function (sys_info = TRUE, platform_info = TRUE, r_version = TRUE, ram = TRUE,
##     cpu = TRUE, byte_compiler = TRUE, linear_algebra = TRUE, locale = TRUE,
##     installed_packages = TRUE, machine = TRUE)
## is_blas_optimize : function (results)
## plot_past : function (test_group, blas_optimize = NULL, cores = 0, log = "y")
## rank_results : function (results, blas_optimize = is_blas_optimize(results), verbose = TRUE)
## upload_results : function (results, url = "http://www.mas.ncl.ac.uk/~ncsg3/form.php", args = NULL,
##     id_prefix = "")
```

```
lsf.str("package:benchmarkmeData")
```

```
## get_datatable_past : function (test_group, blas_optimize = NULL, cores = 0)
## is_blas_optimize : function (results)
## make_data_set : function (from)
## move_files : function (from, to)
## plot_past : function (test_group, blas_optimize = NULL, cores = 0, log = "y")
## select_results : function (test_group, results = NULL, blas_optimize = NULL, cores = 0)
## summarise_results : function (res)
```

```
## This next line takes a lot of time to run, so is commented out here
# benchmark_std()
```

```
get_byte_compiler()
```

```
## byte_optimize
##             2
```

```
gla <- get_linear_algebra()
gsd <- get_sys_details()
```

Calculating the machine precision

Until the IEEE 754 standard (?? Electrical, Committee, and Stevenson (1985)) became available in the 1980s, computers used multiple forms of floating point arithmetic. With some ingenuity, it is possible to detect many aspects of the floating point properties. This is a port(??part) of Mike Malcolm's ENVRON to R. ??ref. It computes the machine precision, radix and number of radix digits.

?? mention Kahan's Paranoia

```
envron <- function(){
D1 <- as.numeric(1) # use variable to avoid constants when double precision invoked
E5 <- 10 # arbitrary scaling for additive equality tests
B9 <- 1E+35 # big number, not necessarily biggest possible
E6 <- 1 # initial value for radix
E9 <- 1 # initial value for machine precision
D0 <- E6
repeat{
  E9 <- E9 / 2 # start of loop to decrease estimated machine precision
  D0 <- E6 + E9 # force storage of sum into a floating-point scalar
  if (D0 <= E6) break
} # repeat reduction while (1+E9) > 1
E9 <- E9 * 2 # restore smallest E9 which gives (1 + E9) > 1
repeat {
   E6 <- E6 + 1 # try different radix values
   D0 <- E6 + E9
   if (D0 <= E6) break
} # until a shift is observed
J1 <- 1 # initial count of radix digits in mantissa
E9 <- 1 # use radix for exact machine precision
repeat {
  E9 <- E9 / E6 # loop while dividing by radix
  J1 <- J1 + 1 # increment counter
  D0 <- D1 + E9 # add tp 1
  if (D0 <= D1) break # test and repeat until equality
}
E9 <- E9 * E6 # recover last value of machine precision
J1 <- J1 - 1 # and adjust the number of digits
mpvals<-list(eps = E9, radix = E6, ndigits = J1)
}
mp <- envron()
E9 <- mp$eps
E6 <- mp$radix
J1 <- mp$ndigits
cat("E9 = ",E9,"  -- the machine precision,  E9=MIN (X  1+X>1)\n")
```

```
## E9 =  2.220446e-16   -- the machine precision,  E9=MIN (X  1+X>1)
```

```
cat( "E9*1E+16=", E9 * 1E+16, "\n")
```

```
## E9*1E+16= 2.220446
```

```
cat( "E6 = ",E6,"  -- the radix of arithmetic", "\n")
```

```
## E6 =  2   -- the radix of arithmetic
```

```
cat( "J1 = ",J1,"  -- the number of radix digits in mantissa of FP numbers", "\n")
```

```
## J1 =  53   -- the number of radix digits in mantissa of FP numbers
```

```
cat( "E6^(-J1+1) * 1E+16=", 1E+16*(E6^(-J1+1)), "\n")
```

```
## E6^(-J1+1) * 1E+16= 2.220446
```

```
cat(".Machine$double.eps=", .Machine$double.eps, "\n")
```

```
## .Machine$double.eps= 2.220446e-16
```

```
cat("From R  .Machine:\n")
```

```
## From R  .Machine:
```

```
cat("double.eps =", .Machine$double.eps, "\n",
    "double.base =", .Machine$double.base, "\n",
    "double.digits =", .Machine$double.digits, "\n" )
```

```
## double.eps = 2.220446e-16
##  double.base = 2
##  double.digits = 53
```

Simple performance examples

There are many performance examples used for testing the speed of computers. A well-known example is the Dongarra, Luszczek, and Petitet (2003) LINPACK test used for ranking supercomputers. These generally employ linear algebra calculations. The example below, by contrast, focuses on the special function computations. Variations on this simple performance test have been used by one of the authors for nearly half a century. Note the use of the `microbenchmark` timing package (Mersmann (2013)).

```
loopesc <- function(nn){
  ss <- 0
  for (i in 1:nn) {
    xx <- exp(sin(cos(1.0*i)))
    ss <- ss + xx
  }
  xx
}
require("microbenchmark")
```

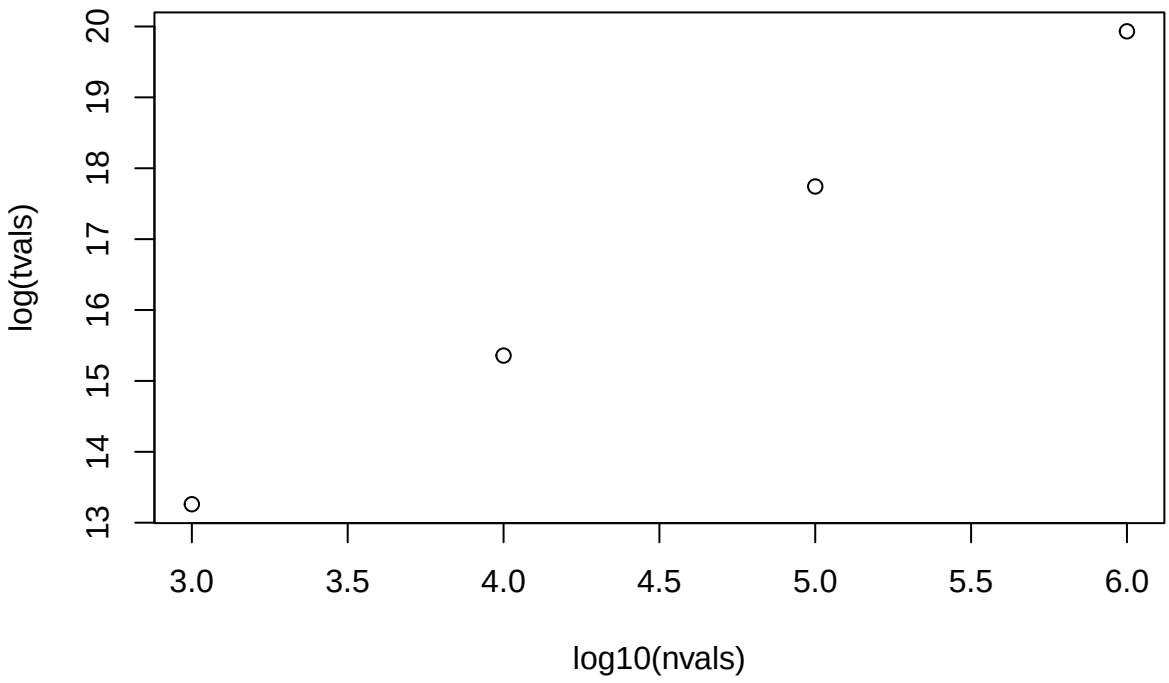
```
## Loading required package: microbenchmark
```

```
nvals<-c(1000, 10000, 100000, 1000000)
tvals<-rep(NA, length(nvals))
i<-0
for (nn in nvals) {
  i <- i + 1
  cat(nn, "\n")
  tt <- microbenchmark(loopesc(nn), unit = 'us')
  print(tt)
```

```
  tvals[i] <- mean(tt$time)
}
```

```
## 1000
## Unit: microseconds
##         expr     min       lq     mean  median      uq      max neval
##  loopesc(nn) 398.001 406.7015 574.2071 425.152 485.951 9526.702   100
## 10000
## Unit: microseconds
##         expr      min     lq     mean   median       uq      max neval
##  loopesc(nn) 3979.301 4038.4 4674.389 4125.051 4465.051 8577.601   100
## 1e+05
## Unit: microseconds
##         expr     min    lq     mean  median      uq     max neval
##  loopesc(nn) 40513.7 42087 50768.37 45719.3 58030.2 76690.1   100
## 1e+06
## Unit: microseconds
##         expr      min     lq     mean median       uq      max neval
##  loopesc(nn) 417113.6 435848 453115.5 444777 461106.5 584429.4   100
```

```
plot(log10(nvals), log(tvals))
```



Sys.info()

```
Sys.info()
```

```
##           sysname           release           version          nodename
##         "Windows"          "10 x64"     "build 22000" "LAPTOP-M87LB32I"
##           machine             login              user    effective_user
##          "x86-64"       "ARKAJYOTI"       "ARKAJYOTI"       "ARKAJYOTI"
```

Issues relating to random number generation

Issues relating to compilation of R

- LAPACK
- BLAS
- others?

Tools for accessing and clearing environments and dataframes

?? should this be here – probably to list things in the workspace

```
sys.frame()
sys.frames()
sys.status()
sys.on.exit()
sys.parents()
sys.calls()
```

Choices

For use in recording tests of R functions and packages for optimization and nonlinear least squares, it seems that the `benchmarkme` function `get_sys_details()` provides more than sufficient information for our needs.

From the above discussion, the following offers a possible compact solution. Users may wish to modify this to their own particular needs.

```
sy <- Sys.info()
tsstr <- format(Sys.time(), "%Y/%m/%d|%H:%M")
cpu <- benchmarkme::get_cpu()
ram <- benchmarkme::get_ram()
machid <- paste(sy["nodename"],":",sy["user"],"-",sy["sysname"],"-",sy["release"],
              "|",cpu$model_name,"|",round(ram/1000^3,2)," GB RAM", sep='')
cat(machid,"\n")
```

```
## LAPTOP-M87LB32I:ARKAJYOTI-Windows-10 x64|AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx|8.59 GB RAM
```

Note that we are reading from the file `MachID.R` so that this file is usable elsewhere and the current document does not become out of sync with the working MachID.

References

Dongarra, J. J., P. Luszczek, and A. Petitet. 2003. "The LINPACK Benchmark: Past, Present, and Future." *Concurrency and Computation: Practice and Experience* 15 (9): 803–20.

Electrical, Institute of, Electronics Engineers. Computer Society. Standards Committee, and David Stevenson. 1985. *IEEE Standard for Binary Floating-Point Arithmetic*. IEEE.

Gillespie, Colin. 2022. *Benchmarkme: Crowd Sourced System Benchmarks*. https://CRAN.R-project.org/package=benchmarkme.

Mersmann, Olaf. 2013. *Microbenchmark: Sub Microsecond Accurate Timing Functions*. http://CRAN.R-project.org/package=microbenchmark.