A Machine Profile Summary

John C. Nash * Arkajyoti Bhattacharjee †

14/06/2021 revised 18/01/2022

Abstract

Many statistical calculations draw on system resources of memory, storage, arithmetic, instructional processing and communications. A given computational environment will have a particular collection of such resources. With workers now commonly working separately and sharing their work virtually, we should consider how to characterize the systems under which calculations are carried out in order that we can diagnose differences in results or performance of the computing methods. Our objective here is to consider how a succinct labeling may provide a **machine profile summary** to assist such analysis and comparisons. To focus our work, we limit our attention to R as the software environment, but the ideas are more generally applicable.

Characterizing a particular computing environment for R

In 2021, we undertook an investigation of tools in R for nonlinear least squares calculations under the Google Summer of Code initiative. In the process, we recognized that we would be comparing calculations performed on **different computing environments** at our disposal. Some of these were based on different hardware, while others were different combinations of given hardware with alternative operating software. In one case a machine failed and had to be replaced, underlining the need to be able to compare results from the defunct system with its replacements.

A useful context for our ideas is provided by the Wikipedia article on Computer Performance (https://en.wikipedia.org/wiki/Computer_performance). We seek a way to provide succinct measures and identifiers of the particular computing environment at hand, leading to a **profile** for a given computing environment for which we will provide a short but meaningful label unique within our project context. This article considers some possibilities available in the R statistical software and language that assist this effort, but the ideas are extensible to other programming systems.

Desired information

There may be multiple terms used to describe some of the information we want, as well as different interpretations of some of the units of measure. For example "Gigabytes" may be 1000^3 or 1024^3 bytes. To settle the ambiguity, the international standard *IEC 80000-13:2008 Quantities and units - Part 13: Information science and technology*(??ref https://en.wikipedia.org/wiki/Gigabyte, https://en.wikipedia.org/wiki/ISO/IEC_80000#Part_13:_Information_science_and_technology) defines 1 Gigabyte (1 GB) as 1000^3 bytes, whereas 1 Gibibyte (1 GiB) as 1024^3 bytes.

Temporal information

For tracking work, it is important 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

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

combination with a machine name and/or other tags. Here is one suggestion based on the R `Sys.time()` function. Moreover, as we were working on opposite sides of the globe, it could be important to either specify time zone or to work with a single "clock" such as UTC (ref?? or expansion?)

```
#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: 2022/06/14|19:45
```

```
# and here is a way to get the UTC time
tutc <- as.POSIXlt(Sys.time(), tz = "UTC")
print(tutc)
```

```
## [1] "2022-06-14 14:15:22 UTC"
```

```
utctstmp <- format(tutc, "%Y/%m/%d|%H:%M")
print(utctstmp)
```

```
## [1] "2022/06/14|14:15"
```

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". Clearly we may want to extend the name if particular software is involved, as follows.
- 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.
- 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, or is using customized packages.

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
```

Machine precision and .Machine

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. In particular, we may be interested in the **machine precision**, that is, the smallest number `eps` such that `1 + eps` is greater than `1`. R performs computations in IEEE "double" precision, and provides the R variable `.Machine` that has "information on the numerical characteristics of the machine R is running on", including the machine precision as `.Machine$double.eps`.

The information in `.Machine` is documented as based on Cody (1988), which is an extension of Mike Malcolm's ENVRON (Malcolm (1972)). A port of this to R is presented below. It computes the machine precision, radix and number of radix digits. A more detailed tool was W. Kahan's PARANOIA (Karpinski (1985)). The rise of IEEE arithmetic has set aside the routine use of such tools, but recent innovations in so-called "half-precision", of which there is more than one variety, may result in renewed interest. See https://en.wikipedia.org/wiki/Half-precision_floating-point_format.

```
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
```

Sys.info()

This R function returns information on the operating system and the generic type of the processor.

```
Sys.info()
```

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

benchmarkme

The CRAN package `benchmarkme` (Gillespie (2022)) provides a wide range of information on system in which it is running, indeed a superset of previous tools mentioned. It is primarily designed to benchmark the capabilities of a particular computing system, so highly appropriate to our current effort, though its output is not succinct. Note that we found a minor bug in the `get_ram()` function of this package for some Windows 10 versions. Our thanks for Colin Gillespie, the author, for making corrections in version 1.0.8, based on communications with one of us (AB).

?? Need to comment on

?? AB: I found the plots of the benchmarks difficult to read. Which end of the plots are "better"?

Some other 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. We used the `microbenchmark` timing package (Mersmann (2013)), which allows us to see the variation in timings, which can be the result of other activities and processes in our computing systems.

Note that it appears that the first cycle of the loop is very slow. ?? is this byte-compiling?

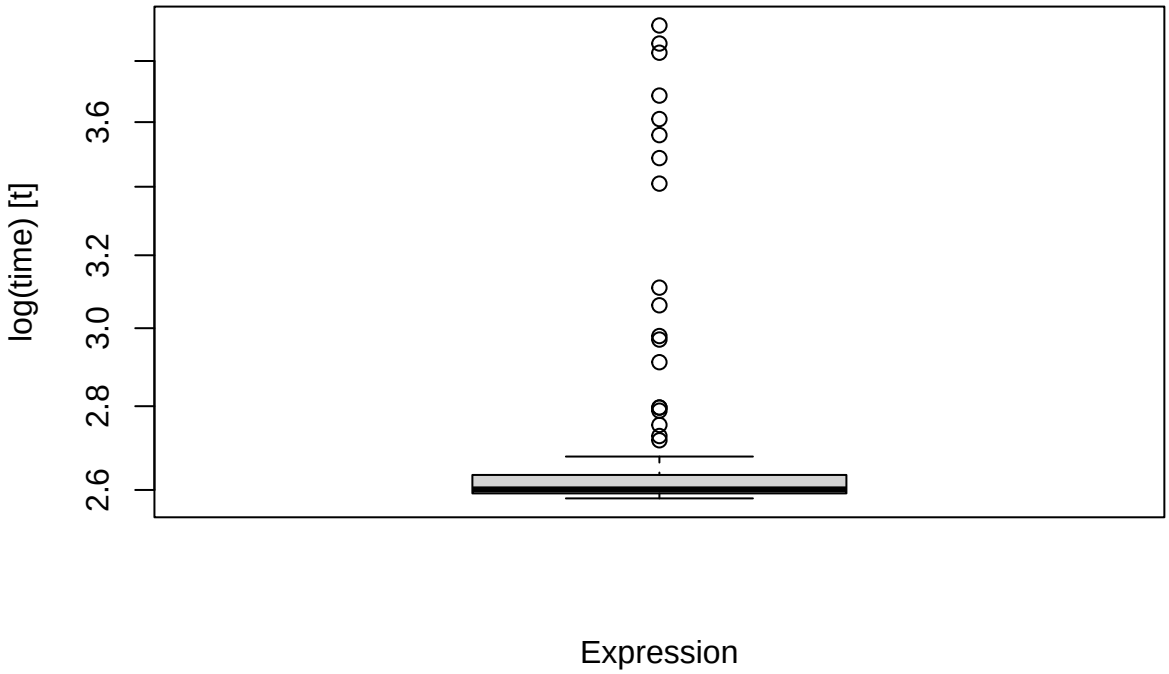
```
loopesc <- function(nn){
  ss <- 0
  for (i in 1:nn) {
    xx <- exp(sin(cos(1.0*i)))
    ss <- ss + xx
  }
  xx
}

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")
  nn <- 10000
  library(compiler)
  le <- cmpfun(loopesc)
  te <- microbenchmark(le(nn), unit='us')
  print(te)
```

```
## Unit: microseconds
##    expr    min     lq     mean median      uq    max neval
##  le(nn) 2580.7 2592.1 2717.662 2601.3 2634.75 3921.3   100
```

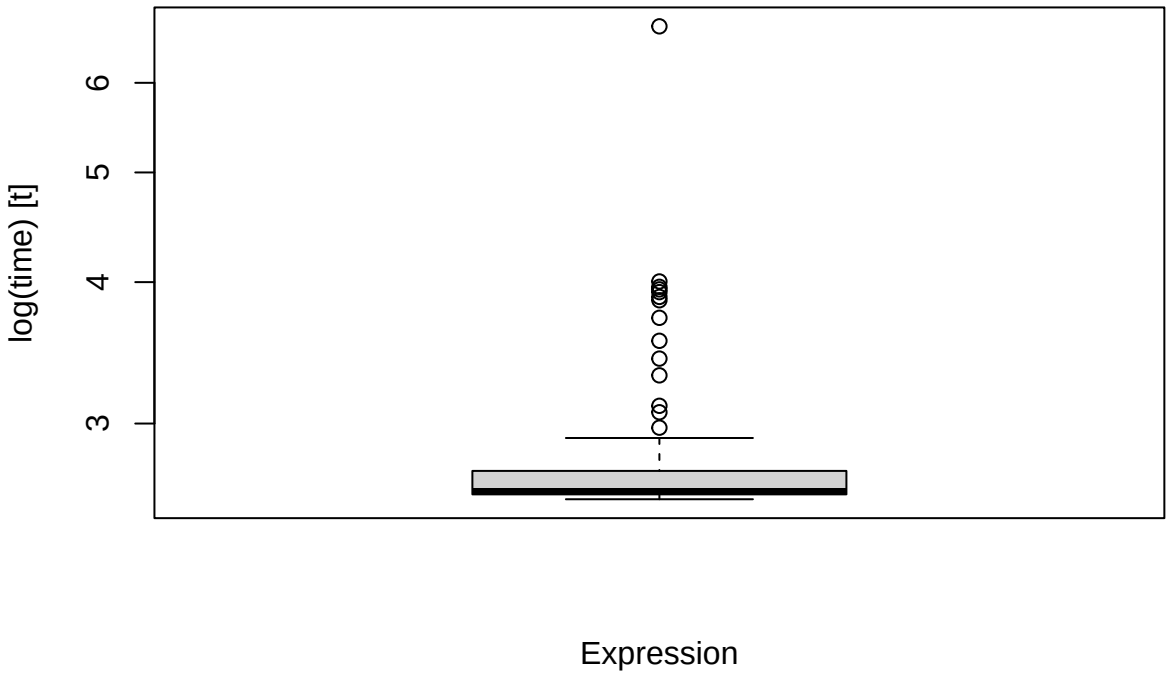
```
  boxplot(te)
```




```
  tt <- microbenchmark(loopesc(nn), unit = 'us')
  print(tt)
```

```
## Unit: microseconds
##         expr    min      lq     mean median      uq    max neval
##  loopesc(nn) 2571.9 2597.55 2802.784 2615.3 2724.55 6730.5   100
```

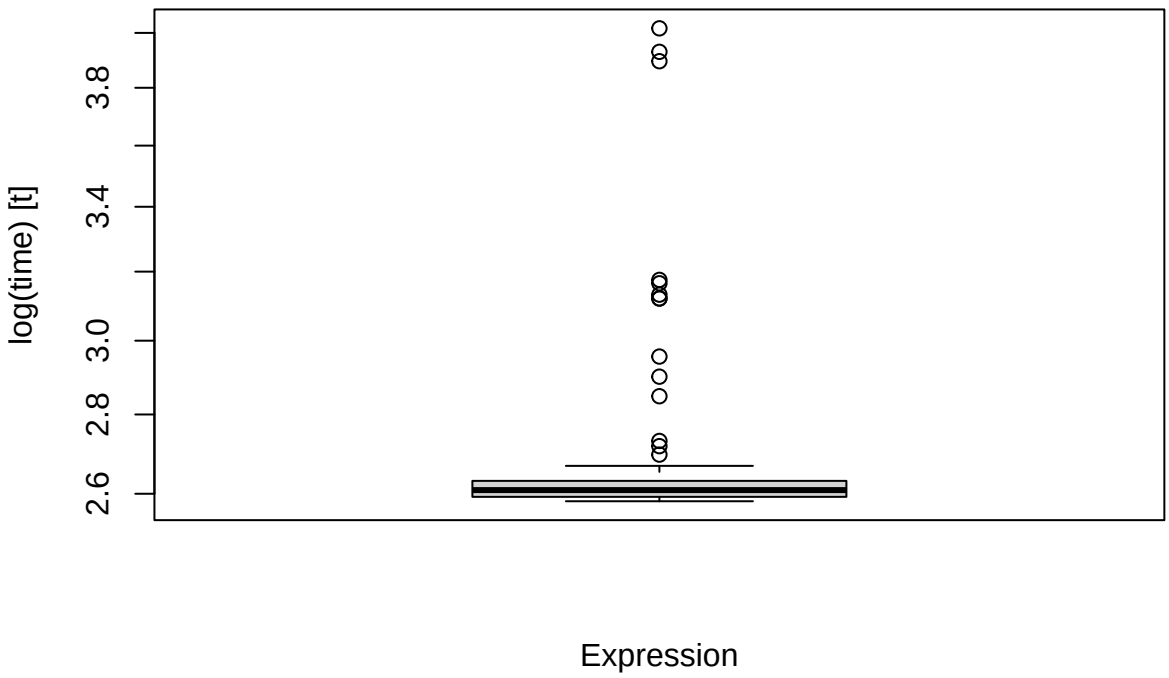
```
  boxplot(tt)
```




```
#  ht<-hist(tt$time)
#  plot(ht, main="Histogram of loopesc times")
  tt2 <- microbenchmark(loopesc(nn), unit = 'us', control=list(warmup=2))
  print(tt2)
```

```
## Unit: microseconds
##         expr    min      lq    mean median     uq    max neval
##  loopesc(nn) 2581.9 2592.55 2687.28 2608.9 2631.5 4017.1   100
```

```
  boxplot(tt2)
```




```
#  ht2<-hist(tt2$time)
#  plot(ht2, main="Histogram of loopesc times after 2 warmup cycles")
# }
# plot(log10(nvals), log(tvals))
```

Missing tools

While we cannot be certain that they do not exist, there do not appear to be tools readily available to allow us to characterize the following features of R computations that might be relevant to understanding machine performance.

- Some users will want to know key information about (pseudo-)random number generation used in their calculations. We suspect that users will need to custom-code ways to get such information about their computations. For example, it is useful to have the specification of the generator and the seed(s) used. For parallel computations, it is important in some applications to use the same set of numbers for each thread, while in others we want to use separate segments of the stream of pseudo-random numbers. From the specification of the generator, it should be possible to know the period (how many numbers are generated before the sequence repeats) and properties concerning correlations between generated numbers. However, in our experience, such information requires a lot of research to uncover.
- R allows for byte compilation of code, and there are several ways in which this can be done. `benchmarkme` returns some information about the general setting used, but details would need user intervention if locally written code is used.
- Compilation of code written in other programming languages and how the resultant routines are called will similarly require user attention. Fortunately, some information is available for heavily used software such as the BLAS and LAPACK, for which `sessionInfo()` reports the libraries used. There are, however, several alternatives, some of which claim high-performance.

A modest profile

For most uses in recording tests of R functions and packages for optimization and nonlinear least squares, which has been our focus, it seems that the `benchmarkme` function `get_sys_details()` provides more than sufficient information for our needs. The following script 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
```

References

Cody, W. J. 1988. "Algorithm 665. MACHAR: A Subroutine to Dynamically Determine Machine Parameters." *ACM Transactions on Mathematical Software* 14 (4): 303–11.

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.

Karpinski, Richard. 1985. "Paranoia: A Floating-Point Benchmark," *Byte Magazine* 10 (2): 223–35.

Malcolm, Michael A. 1972. "Algorithms to Reveal Properties of Floating-Point Arithmetic." *Commun. ACM* 15 (11): 949–51.

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