Package 'aprof'

May 22, 2018

Type Package

Title Amdahl's Profiler, Directed Optimization Made Easy

Version 0.4.1

Date 2018-05-17

Author Marco D. Visser

Maintainer Marco D. Visser <marco.d.visser@gmail.com>

Description Assists the evaluation of whether and
where to focus code optimization, using Amdahl's law and visual aids
based on line profiling. Amdahl's profiler organizes profiling output
files (including memory profiling) in a visually appealing way.
It is meant to help to balance development
vs. execution time by helping to identify the most promising sections
of code to optimize and projecting potential gains. The package is
an addition to R's standard profiling tools and is not a wrapper for them.

License GPL (>= 2)

URL http://github.com/MarcoDVisser/aprof

BugReports http://github.com/MarcoDVisser/aprof/issues

Imports graphics, grDevices, stats, testthat

Repository CRAN

RoxygenNote 5.0.1

NeedsCompilation no

Date/Publication 2018-05-22 18:23:45 UTC

R topics documented:

`aprof` *Create an 'aprof' object for usage in the package 'aprof'*

Description

Create 'aprof' objects for usage with 'aprof' functions

Usage

```
aprof(src = NULL, output = NULL)
```

Arguments

`src`	The name of the source code file (and path if not in the working directory). The source code file is expected to be a a plain text file (e.g. txt, .R), containing the code of the previously profiled program. If left empty, some "aprof" functions (e.g. `readLineDensity`) will attempt to extract this information from the call stack but this is not recommended (as the success of file name detection operations vary). Note that functions that require a defined source file will fail if the source file is not defined or detected in the call stack.
`output`	The file name (and path if not in the working directory) of a previously created profiling exercise.

Details

Creates an "aprof" object from the R-profiler's output and a source file. The objects created through "aprof" can be used by the standard functions plot, summary and print (more specifically: `plot.aprof`, `summary.aprof` and `print.arof`). See the example below for more details.

Using aprof with knitr and within .Rmd or .Rnw documents is not yet supported by the R profiler. Note that setting the chuck option: engine="Rscript", disables line-profiling. Line profiling only works in a interactive session (Oct 2015). In these cases users are advised to use the standard Rprof functions or "profr" (while setting engine="Rscript") and not to rely on line-profiling based packages (for the time being).

Value

An aprof object

Author(s)

Marco D. Visser

See Also

plot.aprof, summary.aprof, print.aprof, Rprof and summaryRprof.

Examples

```
## Not run:
   ## create function to profile
     foo <- function(N){
             preallocate<-numeric(N)
             grow<-NULL
              for(i in 1:N){
                  preallocate[i]<-N/(i+1)
                  grow<-c(grow,N/(i+1))
                 }
     }

     ## save function to a source file and reload
     dump("foo",file="foo.R")
     source("foo.R")

     ## create file to save profiler output
     tmp<-tempfile()

     ## Profile the function
     Rprof(tmp,line.profiling=TRUE)
     foo(1e4)
     Rprof(append=FALSE)

     ## Create a aprof object
     fooaprof<-aprof("foo.R",tmp)
     ## display basic information, summarize and plot the object
     fooaprof
     summary(fooaprof)
     plot(fooaprof)
     profileplot(fooaprof)

     ## To continue with memory profiling:
     ## enable memory.profiling=TRUE
     Rprof(tmp,line.profiling=TRUE,memory.profiling=TRUE)
     foo(1e4)
     Rprof(append=FALSE)
     ## Create a aprof object
     fooaprof<-aprof("foo.R",tmp)
     ## display basic information, and plot memory usage
     fooaprof

     plot(fooaprof)


## End(Not run)
```

`is.aprof` *is.aprof*

Description

Generic lower-level function to test whether an object is an aprof object.

Usage

```
is.aprof(object)
```

Arguments

`object`	Object to test

`plot.aprof` *plot.aprof*

Description

Plot execution time, or total MB usage when memory profiling, per line of code from a previously profiled source file. The plot visually shows bottlenecks in a program's execution time, shown directly next to the code of the source file.

Usage

```
## S3 method for class 'aprof'
plot(x, y, ...)
```

Arguments

`x`	An aprof object as returned by aprof(). If this object contains both memory and time profiling information both will be plotted (as proportions of total time and total memory allocations.
`y`	Unused and ignored at current.
`...`	Additional printing arguments. Unused at current.

Author(s)

Marco D. Visser

Examples

```
## Not run:
# create function to profile
foo <- function(N){
        preallocate<-numeric(N)
        grow<-NULL
         for(i in 1:N){
             preallocate[i]<-N/(i+1)
             grow<-c(grow,N/(i+1))
            }
}

## save function to a source file and reload
dump("foo",file="foo.R")
source("foo.R")

## create file to save profiler output
tmp<-tempfile()

## Profile the function
Rprof(tmp,line.profiling=TRUE)
foo(1e4)
Rprof(append=FALSE)

## Create a aprof object
fooaprof<-aprof("foo.R",tmp)
plot(fooaprof)

## End(Not run)
```

`print.aprof` *Generic print method for aprof objects*

Description

Function that makes a pretty table, and returns some basic information.

Usage

```
## S3 method for class 'aprof'
print(x, ...)
```

Arguments

`x`	An aprof object returned by the function `aprof`.
`...`	Additional printing arguments. Unused.

profileplot *Line progression plot*

Description

A profile plot describing the progression through each code line during the execution of the program.

Usage

```
profileplot(aprofobject)
```

Arguments

aprofobject	An aprof object returned by the function aprof

Details

Given that a source code file was specified in an "aprof" object this function will estimate when each lines was executed. It identifies the largest bottleneck and indicates this on the plot with red markings (y-axis). R uses a statistical profiler which, using system interrupts, temporarily stops execution of a program at fixed intervals. This is a profiling technique that results in samples of "the call stack" every time the system was stopped. The function profileplot uses these samples to reconstruct the progression through the program. Note that the best results are obtained when a decent amount of samples have been taken (relative to the length of the source code). Use print.aprof to see how many samples (termed "Calls") of the call stack were taken.

Author(s)

Marco D. Visser

See Also

plot.aprof

Examples

```
## Not run:
# create function to profile
     foo <- function(N){
             preallocate<-numeric(N)
             grow<-NULL
              for(i in 1:N){
                  preallocate[i]<-N/(i+1)
                  grow<-c(grow,N/(i+1))
                 }
     }

     #save function to a source file and reload
     dump("foo",file="foo.R")
```

```
source("foo.R")

# create file to save profiler output
tmp<-tempfile()

# Profile the function
Rprof(tmp,line.profiling=TRUE)
foo(1e4)
Rprof(append=FALSE)

# Create a aprof object
fooaprof<-aprof("foo.R",tmp)
profileplot(fooaprof)

## End(Not run)
```

---

readLineDensity *readLineDensity*

Description

Reads and calculates the line density (in execution time or memory) of an aprof object returned by the aprof function. If a sourcefile was not specified in the aprof object, then the first file within the profiling information is assumed to be the source.

Usage

```
readLineDensity(aprofobject = NULL, Memprof = FALSE)
```

Arguments

aprofobject	An object returned by aprof, which contains the stack calls sampled by the R profiler.
Memprof	Logical. Should the function return information specific to memory profiling with memory use per line in MB? Otherwise, the default is to return line call density and execution time per line.

Author(s)

Marco D. Visser

summary.aprof *Projected optimization gains using Amdahl's law.*

Description

summary.aprof, projections of code optimization gains.

Usage

```
## S3 method for class 'aprof'
summary(object, ...)
```

Arguments

object	An object returned by the function aprof.
...	Additional [and unused] arguments.

Details

Summarizes an "aprof" object and returns a table with the theoretical maximal improvement in execution time for the entire profiled program when a given line of code is sped-up by a factor (called S in the output). Calculations are done using R's profiler output, and requires line profiling to be switched on. Expected improvements are estimated for the entire program using Amdahl's law (Amdahl 1967), and note that Calculations are subject to the scaling of the problem at profiling. The table output aims to answer whether it is worthwhile to spend hours of time optimizing bits of code (e.g. refactoring in C) and, additionally, identifies where these efforts should be focused. Using aprof one can get estimates of the maximum possible gain. Such considerations are important when one wishes to balance development time vs execution time. All predictions are subject to the scaling of the problem.

Author(s)

Marco D. Visser

References

Amdahl, Gene (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. AFIS Conference Proceedings (30): 483-485.

targetedSummary *targetedSummary*

Description

Allows a detailed look into certain lines of code, which have previously been identified as bottlenecks in combination with a source file.

Usage

```
targetedSummary(target = NULL, aprofobject = NULL, findParent = FALSE)
```

Arguments

target	The specific line of code to take a detailed look at. This can be identified using summary.aprof.
aprofobject	object of class "aprof" returned by the function aprof.
findParent	Logical, should an attempt be made to find the parent of a function call? E.g. "lm" would be a parent call of "lm.fit" or "mean" a parent call of "mean.default". Note that currently, the option only returns the most frequently associated parent call when multiple unique parents exist.

Author(s)

Marco D. Visser

Index