Package 'bigparallelr'

October 12, 2022

Title Easy Parallel Tools

Version 0.3.2

Description Utility functions for easy parallelism in R. Include some reexports from other packages, utility functions for splitting and parallelizing over blocks, and choosing and setting the number of cores used.

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.1

Imports bigassertr (>= 0.1.1), doParallel, flock, parallel, parallelly, RhpcBLASctl

Depends foreach

Suggests testthat, covr

URL https://github.com/privefl/bigparallelr

BugReports https://github.com/privefl/bigparallelr/issues

NeedsCompilation no

Author Florian Privé [aut, cre]

Maintainer Florian Privé <florian.prive.21@gmail.com>

Repository CRAN

Date/Publication 2021-10-02 16:10:02 UTC

R topics documented:

`assert_cores` *Check number of cores*

Description

Check that you are not trying to use too many cores.

Usage

```
assert_cores(ncores)
```

Arguments

`ncores`	Number of cores to check. Make sure is not larger than `getOption("bigstatsr.ncores.max")` (number of logical cores by default). We advise you to use `nb_cores()`. If you really know what you are doing, you can change this default value with `options(bigstatsr.ncores.max = Inf)`.

Details

It also checks if two levels of parallelism are used, i.e. having `ncores` larger than 1, and having a parallel BLAS enabled by default. You could remove this check by setting `options(bigstatsr.check.parallel.blas = FALSE)`.

We instead recommend that you disable parallel BLAS by default by adding `try(bigparallelr::set_blas_ncores(1), silent = TRUE)` to your .Rprofile (**with an empty line at the end of this file**) so that this is set whenever you start a new R session. You can use `usethis::edit_r_profile()` to open your .Rprofile. For this to be effective, you should restart the R session or run `options(default.nproc.blas = NULL)` once in the current session.

Then, in a specific R session, you can set a different number of cores to use for matrix computations using `bigparallelr::set_blas_ncores()`, if you know there is no other level of parallelism involved in your code.

Examples

```
## Not run:

assert_cores(2)

## End(Not run)
```

get_blas_ncores *Number of cores used by BLAS (matrix computations)*

Description

Number of cores used by BLAS (matrix computations)

Usage

```
get_blas_ncores()

set_blas_ncores(ncores)
```

Arguments

ncores	Number of cores to set for BLAS.

Examples

```
get_blas_ncores()
```

nb_cores *Recommended number of cores to use*

Description

This is base on the following rule: use only physical cores and if you have only physical cores, leave one core for the OS/UI.

Usage

```
nb_cores()
```

Value

The recommended number of cores to use.

Examples

```
nb_cores()
```

plus *Add*

Description

Wrapper around Reduce to add multiple arguments. Useful

Usage

```
plus(...)
```

Arguments

...	Multiple arguments to be added together.

Value

```
Reduce('+', list(...))
```

Examples

```
plus(1:3, 4:6, 1:3)
```

register_parallel *Register parallel*

Description

Register parallel in functions. Do makeCluster(), registerDoParallel() and stopCluster() when the function returns.

Usage

```
register_parallel(ncores, ...)
```

Arguments

ncores	Number of cores to use. If using only one, then this function uses foreach::registerDoSEQ().
...	Arguments passed on to makeCluster().

Examples

```
## Not run:

test <- function(ncores) {
  register_parallel(ncores)
  foreach(i = 1:2) %dopar% i
}

test(2)  # only inside the function
foreach(i = 1:2) %dopar% i

## End(Not run)
```

`split_costs` *Split costs in blocks*

Description

Split costs in consecutive blocks using a greedy algorithm that tries to find blocks of even total cost.

Usage

```
split_costs(costs, nb_split)
```

Arguments

`costs`	Vector of costs (e.g. proportional to computation time).
`nb_split`	Number of blocks.

Value

A matrix with 4 columns `lower`, `upper`, `size` and `cost`.

Examples

```
split_costs(costs = 150:1, nb_split = 3)
split_costs(costs = rep(1, 151), nb_split = 3)
split_costs(costs = 150:1, nb_split = 30)
```

split_len *Split length in blocks*

Description

Split length in blocks

Usage

```
split_len(total_len, block_len, nb_split = ceiling(total_len/block_len))
```

Arguments

total_len	Length to split.
block_len	Maximum length of each block.
nb_split	Number of blocks. Default uses the other 2 parameters.

Value

A matrix with 3 columns `lower`, `upper` and `size`.

Examples

```
split_len(10, block_len = 3)
split_len(10, nb_split = 3)
```

split_parapply *Split-parApply-Combine*

Description

A Split-Apply-Combine strategy to parallelize the evaluation of a function.

Usage

```
split_parapply(
  FUN,
  ind,
  ...,
  .combine = NULL,
  ncores = nb_cores(),
  nb_split = ncores,
  opts_cluster = list(),
  .costs = NULL
)
```

Arguments

FUN	The function to be applied to each subset matrix.
ind	Initial vector of indices that will be splitted in nb_split.
...	Extra arguments to be passed to FUN.
.combine	Function to combine the results with do.call. This function should accept multiple arguments (using ...). For example, you can use c, cbind and rbind. This package also provides function plus to add multiple arguments together. The default is NULL, in which case the results are not combined and are returned as a list, each element being the result of a block.
ncores	Number of cores to use. Default uses nb_cores().
nb_split	Number of blocks. Default uses ncores.
opts_cluster	Optional parameters for clusters passed as a named list. E.g., you can use type = "FORK" to use forks instead of clusters. You can also use outfile = "" to redirect printing to the console.
.costs	Vector of costs (e.g. proportional to computation time) associated with each element of ind. Default is NULL (same cost).

Details

This function splits indices in parts, then apply a given function to each part and finally combine the results.

Value

Return a list of ncores elements, each element being the result of one of the cores, computed on a block. The elements of this list are then combined with do.call(.combine, .) if .combined is not NULL.

Examples

```
## Not run:

str(
  split_parapply(function(ind) {
    sqrt(ind)
  }, ind = 1:10000, ncores = 2)
)

## End(Not run)
```

split_vec *Split object in blocks*

Description

Split object in blocks

Usage

```
split_vec(x, block_len, nb_split = ceiling(length(x)/block_len))

split_df(df, block_len, nb_split = ceiling(nrow(df)/block_len))
```

Arguments

x	Vector to be divided into groups.
block_len	Maximum length (or number of rows) of each block.
nb_split	Number of blocks. Default uses the other 2 parameters.
df	Data frame to be divided into groups.

Value

A list with the splitted objects.

Examples

```
split_vec(1:10, block_len = 3)
str(split_df(iris, nb_split = 3))
```

Index