

Package ‘rparallel’

April 9, 2008

Version 0.6-10

Date 2008-04-10

Title R/parallel : Accessible Parallel Computing for Desktop Computers in R

Author Gonzalo Vera Rodríguez <gonzalo.vera.rodriguez@gmail.com>

Maintainer Gonzalo Vera Rodríguez <gonzalo.vera.rodriguez@gmail.com>

Description R/parallel is an R add-on package that easily and effectively enables the automatic parallelization of loops without data dependencies.

Depends R (>= 2.6.1)

License GPL version 2 or newer for non-commercial (TODO)

URL <http://www.rparallel.org>

R topics documented:

runParallel	1
Index	5

runParallel	<i>Accessible Parallel Computing for Desktop Computers in R</i>
-------------	---

Description

The package R/parallel enables the parallel execution of loops without data dependencies just by adding a single function: runParallel.

Enclosing this function and the target loop within an if/else control structure, R/parallel is able to locate the loop and automate its parallelization.

Usage

```
runParallel( resultVar, resultOp, cpuLimit, nWorkers, verbose, tempDir )
```

Arguments

resultVar	Vector of character strings with the variable names which values have to be retrieved during the loop iterations to <i>reduce</i> its final value (i.e. to obtain their correct value as if they were being run sequentially).
resultOp	Vector of character strings with the operations or functions like <code>+</code> , <code>average</code> , <code>max</code> or <code>rbind</code> , that have to be applied each iteration to each <code>resultVar[]</code> variable, .
cpuLimit	(optional) numeric indicating the maximum percentage of processor usage value allowed to be used by <code>R/parallel</code> . This value is checked by each worker before the beginning of each iteration. During the calculation the processor can reach 100%. In this situations, the processor usage, as long as <code>R/parallel</code> is not preemptive, is regulated at beginning of the next iteration. Valid values are numerics from 1 to 100. Default value is 100 (i.e. no processor control).
nWorkers	(optional) numeric indicating the number of new R instances (i.e. additional processes) that will run the workers. The problem size (i.e. the number of iterations) is split among all the workers. Setting more workers than processor units (e.g. cores) usually does not improve the performance. However, when iterating through long vectors that will be split (i.e. thousands elements), setting more workers can help to speed up the calculation. Default value is 2.
verbose	(optional) character string indicating the verbosity level of <code>R/parallel</code> . Valid values are: "silent" : No output, "normal" : Minimum information, "info" : Additional information, including elapsed time, and, "debug" : Highest amount of information, only for internal developers. Default value is "normal".
tempDir	(optional) character string indicating the "PATH" to a folder with enough free space to store temporary files used during calculation. If not provided, the R temporary directory, as returned by <code>tempdir()</code> , will be used. Default value is NULL.

Details

`runParallel` requests to the `R/parallel` internal utilities to retrieve the expression enclosed inside the following `else` section of code (i.e. the 'for' loop to be parallelized). `R/parallel`, combining the information retrieve from the current execution with the information provided by the user, automates the parallel execution of the loop. Once the parallel execution has finished in the workers it updates the variables indicated by the user.

Value

It returns no value. However it updates the values of the variables indicated within the argument vector `resultVar`.

Application

The following pseudo-R-code simulates the typical layout of a function which includes a central loop that iterates through one of his arguments to perform a calculation:

```

myFunction <- function( arg1, arg2, ... )
{
  # Initial sequence of statements
  # Initializing variables and checking arguments
  variable1 <- constant1
  variable2 <- otherFunction( arg1 )
  ...

  # loop
  for( index in FirstValue:LastValue )
  {
    (more statements/loops/expressions/function calls/etc)
    ...
    tempVar <- functionA( arg1[ index ], variable1, ... )
    tempVar <- functionB( arg2[ index ], variable1, ... )
    resultVar1 <- resultOP1( resultVar1, tempVar1)
    resultVar2 <- resultOP2( resultVar2, tempVar2)
  }

  # Finalizing calculation. Final sequence of statements
  (more statements/expressions/function calls/etc)
  ...

  return( anyCalculatedValue )
}

```

To include R/parallel, an `if{} else{} control structure` has to be added as shows the following example:

```

myFunction <- function( arg1, arg2, ... )
{
  # Initial sequence of statements
  # Initializing variables and checking arguments
  variable1 <- constant1
  variable2 <- otherFunction( arg1 )
  ...
  if( "rparallel" %in% names( getLoadedDLLs() ) )
  {
    runParallel( resultVar=c("resultVar1", "resultVar2" ),
                resultOp= c("resultOP1", "resultOP2" ) )
  }
  else
  {
    # loop
    for( index in FirstValue:LastValue )
    {
      (more statements/loops/expressions/function calls/etc)
      ...
      tempVar1 <- functionA( arg1[ index ], variable1, ... )
      tempVar2 <- functionB( arg2[ index ], variable1, ... )
      resultVar1 <- resultOP1( resultVar1, tempVar1)
      resultVar2 <- resultOP2( resultVar2, tempVar2)
    }
  }
}

```

```

}

# Finalizing calculation. Final sequence of statements
# (more statements/expressions/function calls/etc)
...

return( anyCalculatedValue )
}

```

Note

A data dependency occurs when the calculation of a value depends on the result of a previous iteration (e.g. $a[n] \leftarrow a[n-1] + 1$). Results when running with data dependencies are unpredictable.

Author(s)

Gonzalo Vera Rodríguez <gonzalo.vera.rodriguez@gmail.com>

Examples

```

## Not run:
# 1. Adapt your function, for example:
qtlMapping <- function( map, genotypes, traits )
{
  result <- NULL
  if( "rparallel" %in% names( getLoadedDLLs() ) )
  {
    runParallel( resultVar="result", resultOp="rbind", nWorkers=4 )
  }
  else
  {
    for( idx in 1:nrow( traits ) )
    {
      tmpResult          <- MQM(map, genotypes, traits[idx,])
      rownames( tmpResult ) <- rownames(traits)[idx]
      result             <- rbind( result, tmpResult )
    }

    return( result )
  }
}
#2. Load the library
library(rparallel)

#3. Run your function as usual!
myQTLmap <- qtlMapping( map, genotypes, traits )

## End(Not run)

```

Index

*Topic **method**

`runParallel, 1`

`rparallel (runParallel), 1`

`runParallel, 1`