Package 'BiocParallel'

February 7, 2015

Type Package

Title Bioconductor facilities for parallel evaluation

Version 1.0.3

Description This package provides modified versions and novel implementation of functions for parallel evaluation, tailored to use with Bioconductor objects.

biocViews Infrastructure

License GPL-2 | GPL-3

Depends methods

Imports parallel, foreach, tools, BatchJobs, BBmisc, BiocGenerics

Suggests doParallel, snow, Rmpi, GenomicRanges, RNAseqData.HNRNPC.bam.chr14, Rsamtools, GenomicAlignments, ShortRead, RUnit, BiocStyle, knitr

Collate.unix AllGenerics.R BiocParallelParam-class.R ErrorHandling.R bpbackend-methods.R bpisup-methods.R bplapply-methods.R bpmapply-methods.R bpiterate-methods.R bpschedule-methods.R bpstart-methods.R bpstop-methods.R bpvec-methods.R bpvectorize-methods.R bpworkers-methods.R bpaggregate-methods.R DoparParam-class.R MulticoreParam-class.R register.R SerialParam-class.R SnowParam-class.R BatchJobsParam-class.R utilities.R unix/mclapply.R unix/pvec.R unix/bpiterate.R unix/zzz.R

Collate.windows AllGenerics.R BiocParallelParam-class.R ErrorHandling.R bpbackend-methods.R bpisup-methods.R bplapply-methods.R bpmapply-methods.R bpiterate-methods.R bpschedule-methods.R bpstart-methods.R bpstop-methods.R bpvec-methods.R bpvectorize-methods.R bpworkers-methods.R bpaggregate-methods.R DoparParam-class.R MulticoreParam-class.R register.R SerialParam-class.R SnowParam-class.R BatchJobsParam-class.R utilities.R windows/zzz.R

VignetteBuilder knitr

R topics documented:

`BiocParallel-package` *Bioconductor facilities for parallel evaluation*

Description

This package provides modified versions and novel implementation of functions for parallel evaluation, tailored to use with Bioconductor objects.

Details

This package uses code from the `parallel` package,

Author(s)

Author: Bioconductor Package Maintainer [cre], Martin Morgan [aut], Michel Lang [aut], Ryan Thompson [aut]

Maintainer: Bioconductor Package Maintainer <maintainer@bioconductor.org>

BatchJobsParam-class *Enable parallelization on batch systems*

Description

This class is used to parameterize scheduler options on managed high-performance computing clusters.

Usage

```
BatchJobsParam(workers, catch.errors = TRUE, cleanup = TRUE,
    work.dir = getwd(), stop.on.error = FALSE, seed = NULL,
    resources = NULL, conffile = NULL, cluster.functions = NULL,
    progressbar = TRUE, ...)
```

Arguments

workers	integer(1)Number of workers to divide tasks (e.g., elements in the first argument of bplapply) between. On Multicore and SSH backends, this defaults to all available nodes. On managed (e.g., slurm, SGE) clusters workers defaults to NA, meaning that the number of workers equals the number of tasks. See argument n.chunks in chunk and submitJobs for more information.
catch.errors	logical(1)Flag to determine in apply-like functions (see e.g. bplapply) whether to quit with an error as soon as one application fails or encapsulation of function calls in try blocks which triggers a resume mechanism (see bpresume). Defaults to TRUE.
cleanup	logical(1)BatchJobs creates temporary directories in the work.dir. If cleanup is set to TRUE (default), the directories are removed from the file systems automatically. Set this to FALSE whenever it might become necessary to utilize any special functionality provided by BatchJobs. To retrieve the registry, call loadRegistry on the temporary directory.
work.dir	character(1)Directory to store temporary files. Note that this must be shared across computational nodes if you use a distributed computing backend. Default ist the current working directory of R, see getwd.
stop.on.error	logical(1)Stop all jobs as soon as one jobs fails (stop.on.error == TRUE) or wait for all jobs to terminate. Default is FALSE.
seed	integer(1L)Set an initial seed for the RNG. See makeRegistry for more information. Default is NULL where a random seed is chosen upon initialization.
resources	list()List of job specific resources passed to submitJobs. Default is NULL where the resources defined in the configuration are used.
conffile	character(1)URI to a custom BatchJobs configuration file used for execution. Default is NULL which relies on BatchJobs to handle configuration files.
cluster.functions	ClusterFunctionsSpecify a specific cluster backend using on of the constructors provided by BatchJobs, see ClusterFunctions. Default is NULL where the default cluster functions defined in the configuration are used.

progressbar	logical(1) Suppress the progress bar used in BatchJobs and be less verbose. Default is FALSE.
...	Addition arguments, currently not handled.

BatchJobsParam constructor

Return an object with specified values. The object may be saved to disk or reused within a session.

Methods

The following generics are implemented and perform as documented on the corresponding help page: bpworkers, bpstart, bpstop, bpisup, bpbackend, bpbackend<-

Author(s)

Michel Lang, mailto:michellang@gmail.com

See Also

getClass("BiocParallelParam") for additional parameter classes.

register for registering parameter classes for use in parallel evaluation.

Examples

```
p <- BatchJobsParam(progressbar=FALSE)
bplapply(1:10, sqrt, BPPARAM=p)

## Not run:
## see vignette for additional explanation
funs <- makeClusterFunctionsSLURM("~/slurm.tmpl")
param <- BatchJobsParam(4, cluster.functions=funs)
register(param)
bplapply(1:10, function(i) sqrt)

## End(Not run)
```

BiocParallelParam-class

Virtual classes (for developer reference)

Description

These classes are primarily relevant to developers.

BiocParallelParam is a virtual class on which all parameter classes extend. It has no slots.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org

Examples

```
getClass("BiocParallelParam")
```

---

bpaggregate *Apply a function on subsets of data frames*

Description

This is a parallel version of aggregate.

Usage

```
## S4 method for signature formula,BiocParallelParam
bpaggregate(x, data, FUN, ..., BPPARAM=bpparam())

## S4 method for signature data.frame,BiocParallelParam
bpaggregate(x, by, FUN, ..., simplify=TRUE, BPPARAM=bpparam())

## S4 method for signature matrix,BiocParallelParam
bpaggregate(x, by, FUN, ..., simplify=TRUE, BPPARAM=bpparam())

## S4 method for signature ANY,missing
bpaggregate(x, ..., BPPARAM=bpparam())
```

Arguments

x	A data.frame, matrix or a formula.
by	A list of factors by which x is split; applicable when x is data.frame or matrix.
data	A data.frame; applicable when x is a formula.
FUN	Function to apply.
...	Additional arguments for FUN.
simplify	If set to TRUE, the return values of FUN will be simplified using simplify2array.
BPPARAM	An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation.

Details

bpaggregate is a generic with methods for data.frame matrix and formula objects. x is divided into subsets according to factors in by. Data chunks are sent to the workers, FUN is applied and results are returned as a data.frame.

The function is similar in spirit to aggregate from the stats package but aggregate is not explicitly called. The bpaggregate formula method reformulates the call and dispatches to the data.frame method which in turn distributes data chunks to workers with bplapply.

Value

See aggregate.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org.

Examples

```
if (all(require(Rsamtools) &&
        require(GenomicAlignments))) {

  fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
  param <- ScanBamParam(what = c("flag", "mapq"))
  gal <- readGAlignments(fl, param=param)

  ## Report the mean map quality by range cutoff:
  cutoff <- rep(0, length(gal))
  cutoff[start(gal) > 1000 & start(gal) < 1500] <- 1
  cutoff[start(gal) > 1500] <- 2
  bpaggregate(as.data.frame(mcols(gal)$mapq), list(cutoff = cutoff), mean)

}
```

---

bpcontrols *Control (start, stop, query) back-end Params*

Description

Use functions on this page to start, stop, and query the 'back-ends' that perform a parallel evaluation.

Usage

```
bpworkers(x, ...)

bpstart(x, ...)
bpstop(x, ...)
bpisup(x, ...)

bpbackend(x, ...)
bpbackend(x, ...) <- value
```

Arguments

x	An instance of a BiocParallelParam class, e.g., MulticoreParam, SnowParam, DoparParam. x can be missing, in which case the default back-end (see register) is used.
...	Additional arguments, perhaps used by methods.
value	Replace value for back-end.

Details

bpworkers reports the number of workers in the back-end as a scalar integer with value >= 0.

bpstart starts the back-end, if necessary. For instance, MulticoreParam back-ends do not need to be started, but SnowParam back-ends do. bp* functions like bplapply will automatically start the back-end if necessary.

bpstop stops the back-end, if necessary and possible.

bpisup tests whether the back-end is available for processing, returning a scalar logical value.

bpbackend retrieves an object representing the back end, if possible. Not all back-ends can be retrieved; see showMethods("backend").

bpbackend<- updates the back end, and is only meant for developer use.

Value

bpworkers returns a scalar integer >= 0.

bpisup returns a scalar logical.

bpstart, bpstop return an updated x, invisibly.

bpbackend, bpbackend<- return or accept back end-specific objects.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org.

See Also

BiocParallelParam for possible values of x.

Examples

```
bpworkers(SerialParam())

## Not run:
  p <- SnowParam(2L)
  bpworkers(p)          # 2 local nodes, communicating via sockets
  bpstart(p)            # start cluster
  bplapply(1:10, sqrt, BPPARAM=p)
  bpstop(p)             # stop cluster

  p <- SnowParam(4L)
  bplapply(1:10, sqrt, BPPARAM=p)  # automatically start / stop cluster
```

```
## End(Not run)
```

---

bpiterate *Parallel iteration over an indeterminate number of data chunks*

Description

bpiterate iterates over an indeterminate number of data chunks (e.g., records in a file). Each chunk is processed by parallel workers in an asynchronous fashion; as each worker finishes it recieves a new chunk. Data are traversed a single time.

Usage

```
bpiterate(ITER, FUN, ..., BPPARAM=bpparam())

## S4 method for signature ANY,ANY,missing
bpiterate(ITER, FUN, ..., BPPARAM=bpparam())

## S4 method for signature ANY,ANY,BiocParallelParam
bpiterate(ITER, FUN, ..., BPPARAM=bpparam())
```

Arguments

ITER	A function with no arguments that returns an object to process, generally a chunk of data from a file. When no objects are left (i.e., end of file) it should return NULL and continue to return NULL reguardless of the number of times it is invoked after reaching the end of file. This function is run on the master.
FUN	A function to process the object returned by ITER; run on parallel workers separate from the master. When BPPARAM is a MulticoreParam, FUN is 'decorated' with additional arguments and therefore must have ... in the signature.
BPPARAM	An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation, or a list of BiocParallelParam instances, to be applied in sequence for nested calls to bpiterate. Currently only MulticoreParam is supported for bpiterate.
...	Arguments to other methods, specifically named arguments for FUN, or REDUCE or init. • REDUCE: Optional function that combines (reduces) output from FUN. As each worker returns, the data are combined with the REDUCE function. REDUCE takes 2 arguments; one is the current result and the other is the output of FUN from a worker that just finished. Currently not supported on Windows. • init: Optional initial value for REDUCE; must be of the same type as the object returned from FUN. When supplied, reduce.in.order is set to TRUE. • reduce.in.order: Logical. When TRUE, REDUCE is applied to the results from the workers in the same order the tasks were sent out.

Details

Currently only MulticoreParam and SerialParam are supported.

`bpiterate` iterates through an unknown number of data chunks, dispatching chunks to parallel workers as they become available. In contrast, other `bp*apply` functions such as codebplapply or `bpmapply` require the number of data chunks to be specified ahead of time. This quality makes `bpiterate` useful for iterating through files of unknown length.

`ITER` serves up chunks of data until the end of the file is reached at which point it returns NULL. Note that `ITER` should continue to return NULL reguardless of the number of times it is invoked after reaching the end of the file. `FUN` is applied to each object (data chunk) returned by `ITER`.

Value

A `list` of output type specified by `FUN`. When `REDUCE` is supplied the list is of length 1.

Author(s)

Valerie Obenchain mailto:vobencha@fhcrc.org.

The multi-core implementation is a modification of the `sclapply` in `HTSeqGeni` by Gregorie Pau.

See Also

- `bpvec` for parallel, vectorized calculations.
- `bplapply` for parallel, lapply-like calculations.
- `BiocParallelParam` for details of `BPPARAM`.

Examples

```
if (all(require(Rsamtools) &&
        require(RNAseqData.HNRNPC.bam.chr14) &&
        require(GenomicAlignments) &&
        require(ShortRead) &&
        .Platform$OS.type != "windows")) {

  ## ------------------------------------------------------------------------
  ## Iterating through a BAM file
  ## ------------------------------------------------------------------------

  ## Select a single file and set yieldSize in the BamFile object.
  fl <- RNAseqData.HNRNPC.bam.chr14_BAMFILES[[1]]
  bf <- BamFile(fl, yieldSize = 300000)

  ## bamIterator() is initalized with a BAM file and returns a function.
  ## The return function requires no arguments and iterates through the
  ## file returning data chunks the size of yieldSize.
  bamIterator <- function(bf) {
      done <- FALSE
      if (!isOpen( bf))
          open(bf)
```

```
    function() {
        if (done)
            return(NULL)
        yld <- readGAlignments(bf)
        if (length(yld) == 0L) {
            close(bf)
            done <<- TRUE
            NULL
        } else yld
    }
}

## Initalize the iterator.
ITER <- bamIterator(bf)

## Create a FUN that counts reads in a region of interest.
roi <- GRanges("chr14", IRanges(seq(19e6, 107e6, by = 10e6), width = 10e6))
counter <- function(reads, roi, ...) {
    countOverlaps(query = roi, subject = reads)
}
## Create a MulticoreParam and call bpiterate().
bpparam <- MulticoreParam(workers = 2)
res <- bpiterate(ITER, counter, BPPARAM = bpparam, roi = roi)

## The result length is the same as the number of data chunks.
length(res)
colSums(do.call(rbind, res))

## ------------------------------------------------------------------------
## Iterating through a FASTA file
## ------------------------------------------------------------------------

## Set data chunk size with n in the FastqStreamer object.
sp <- SolexaPath(system.file(extdata, package = ShortRead))
fl <- file.path(analysisPath(sp), "s_1_sequence.txt")
fqs <- FastqStreamer(fl, n = 100)

## Create an iterator that returns data chunks the size of n.
fastqIterator <- function(fqs) {
    done <- FALSE
    if (!isOpen(fqs))
        open(fqs)

    function() {
        if (done)
            return(NULL)
        yld <- yield(fqs)
        if (length(yld) == 0L) {
            close(fqs)
            done <<- TRUE
            NULL
        } else yld
```

```
    }
}

## Initialize the iterator.
ITER <- fastqIterator(fqs)

## The processor summarizes the number of times each sequence occurs.
summary <- function(reads, ...) {
     tables(reads, n = 0)$distribution
}

bpparam <- MulticoreParam(workers = 2)
bpiterate(ITER, summary, BPPARAM = bpparam)

## Results from the workers are combined on the fly when a
## REDUCE function is provided. Collapsing the data in this
## way can substantially reduce memory requirements.
fqs <- FastqStreamer(fl, n = 100)
ITER <- fastqIterator(fqs)
bpiterate(ITER, summary, BPPARAM = bpparam, REDUCE = merge, all = TRUE)

## ----------------------------------------------------------------------
## Multiple files
## ----------------------------------------------------------------------
## Currently bpiterate() is only implemented for the multi-core
## environment (i.e., MulticoreParam). A single machine may not
## have enough memory to process multiple files. In this case the
## files can be distributed over a snow cluster with bplapply()
## then processed with bpiterate() on each cluster node.

## Select a subset of files, create a BamFileList.
fls <- RNAseqData.HNRNPC.bam.chr14_BAMFILES[1:3]
bfl <- BamFileList(fls, yieldSize = 200000)

## Cluster size is defined by the number of files.
snowp <- SnowParam(workers = length(bfl))

## Currently bpiterate() is only supported in the multi-core environment
## and must use MulticoreParam.
myFUN <- function(file, bamIterator, counter, ...) {
  library(BiocParallel)  ## for bpiterate
  library(Rsamtools)     ## for Bam file manipulation
  library(GenomicAlignments) ## for readGAlignments
  ITER <- bamIterator(file)
  bpiterate(ITER, counter, BPPARAM = MulticoreParam(workers=2), roi = roi)
}

## Distribute the files across the snow cluster workers with bplapply().
## Each cluster node acts as a master and spawns 2 children.
res <- bplapply(bfl, myFUN, BPPARM = snowp, bamIterator = bamIterator,
                counter = counter, roi = roi)

## The result is of length 3 (# of files).
```

```
  ## > length(res)
  ## [1] 3


  ## Each list element is of length 5 (# of iterations of size yieldSize).
  ## >  elementLengths(res)
  ## ERR127306 ERR127307 ERR127308
  ##         5         5         5
}
```

---

bplapply *Parallel lapply-like functionality*

Description

bplapply applies FUN to each element of X. Any type of object X is allowed, provided length, [, and [[methods are available. The return value is a list of length equal to X, as with lapply.

Usage

```
bplapply(X, FUN, ..., BPRESUME = getOption("BiocParallel.BPRESUME", FALSE),
    BPPARAM=bpparam())

## S4 method for signature ANY,ANY
bplapply(X, FUN, ..., BPRESUME = getOption("BiocParallel.BPRESUME", FALSE),
    BPPARAM=bpparam())

## S4 method for signature ANY,missing
bplapply(X, FUN, ..., BPRESUME = getOption("BiocParallel.BPRESUME", FALSE),
    BPPARAM=bpparam())

## S4 method for signature ANY,BiocParallelParam
bplapply(X, FUN, ..., BPRESUME = getOption("BiocParallel.BPRESUME", FALSE),
    BPPARAM=bpparam())
```

Arguments

X	Any object for which methods length, [, and [[are implemented.
FUN	The function to be applied to each element of X.
...	Additional arguments for FUN, as in lapply
BPRESUME	Flag to determine if a previous partially successful run should be resumed. See bpresume for details.
BPPARAM	An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation, or a list of BiocParallelParam instances, to be applied in sequence for nested calls to bplapply.

Details

See showMethods{bplapply} for additional methods, e.g., method?bplapply("MulticoreParam").

Value

See lapply.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org. Original code as attributed in mclapply.

See Also

bpvec for parallel, vectorized calculations.

BiocParallelParam for possible values of BPPARAM.

Examples

```
showMethods("bplapply")

## ten tasks (1:10) so ten calls to FUN default registered parallel
## back-end. Compare with bpvec.
system.time(result <- bplapply(1:10, function(v) {
    message("working") ## 10 tasks
    sqrt(v)
}))
result
```

---

bpmapply *Parallel mapply-like functionality*

Description

bpmapply applies FUN to first elements of ..., the second elements and so on. Any type of object in ... is allowed, provided length, [, and [[methods are available. The return value is a list of length equal to the length of all objects provided, as with mapply.

Usage

```
bpmapply(FUN, ..., MoreArgs=NULL, SIMPLIFY=TRUE, USE.NAMES=TRUE,
    BPRESUME=getOption("BiocParallel.BPRESUME", FALSE), BPPARAM=bpparam())

## S4 method for signature ANY,ANY
bpmapply(FUN, ..., MoreArgs=NULL, SIMPLIFY=TRUE, USE.NAMES=TRUE,
    BPRESUME=getOption("BiocParallel.BPRESUME", FALSE), BPPARAM=bpparam())

## S4 method for signature ANY,missing
```

```
bpmapply(FUN, ..., MoreArgs=NULL, SIMPLIFY=TRUE, USE.NAMES=TRUE,
    BPRESUME=getOption("BiocParallel.BPRESUME", FALSE), BPPARAM=bpparam())

## S4 method for signature ANY,BiocParallelParam
bpmapply(FUN, ..., MoreArgs=NULL, SIMPLIFY=TRUE, USE.NAMES=TRUE,
    BPRESUME=getOption("BiocParallel.BPRESUME", FALSE), BPPARAM=bpparam())
```

Arguments

FUN	The function to be applied to each element passed via ....
...	Objects for which methods length, [, and [[are implemented. All objects must have the same length or shorter objects will be replicated to have length equal to the longest.
MoreArgs	List of additional arguments to FUN.
SIMPLIFY	If TRUE the result will be simplified using simplify2array.
USE.NAMES	If TRUE the result will be named.
BPRESUME	Flag to determine if a previous partially successful run should be resumed. See bpresume for details.
BPPARAM	An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation.

Details

See showMethods{bplapply} for additional methods, e.g., method?bplapply("MulticoreParam").

Value

See lapply.

Author(s)

Michel Lang . Original code as attributed in mclapply.

See Also

bpvec for parallel, vectorized calculations.

BiocParallelParam for possible values of BPPARAM.

Examples

```
showMethods("bpmapply")

## ten tasks (1:10) so ten calls to FUN default registered parallel
## back-end. Compare with bpvec.
result <- bpmapply(function(greet, who) {
    paste(Sys.getpid(), greet, who)
}, c("morning", "night"), c("sun", "moon"))
cat(paste(result, collapse="\n"), "\n")
```

bpresume *Resume computation with partial results*

Description

Resume partial successful calls to bplapply or bpmapply

Usage

```
bplasterror()
bpresume(expr)
```

Arguments

expr	expression A expression which calls either bplapply or bpmapply.

Usage

The resume mechanism is triggered if the argument catch.errors of the BiocParallelParam class is set to TRUE. The methods bplapply and bpmapply then store the current state of computation. Recalling the call directory with argument BPRESUME set to TRUE will then only compute the missing parts of the previous call and merge the results.

Alternatively, if the call to bplapply and bpmapply is inside a function not accessible directly by the user, the last call can be embedded into bpresume which sets an option accordingly to enable the resume feature down in the call stack.

The function bplasterror returns a LastError object containing the partial results and errors to investigate.

Note that nested calls of the apply functions can cause troubles in some scenarios.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org

Examples

```
## ------------------------------------------------------------------------
## Catch errors:
## ------------------------------------------------------------------------

## By default catch.errors is TRUE in BiocParallelParam objects.
SnowParam(workers = 2)

## If catch.errors is FALSE an ill-fated bplapply() simply stops
## displaying the error message.
snowp <- SnowParam(workers = 2, catch.errors = FALSE)
## Not run:
> res <- bplapply(list(1, "two", 3), sqrt, BPPARAM = snowp)
```

```
Error in checkForRemoteErrors(val) :
  one node produced an error: non-numeric argument to mathematical function

## End(Not run)

## When catch.errors is TRUE the same call provides traceback
## information (truncated here) and suggests use of bplasterror() and
## bpresume().
snowp <- SnowParam(workers = 2)
## Not run:
> res <- bplapply(list(1, "two", 3), sqrt, BPPARAM = snowp)
Error: 1 errors; first error:
  Error in FUN(...): non-numeric argument to mathematical function

For more information, use bplasterror(). To resume calculation, re-call
  the function and set the argument BPRESUME to TRUE or wrap the
  previous call in bpresume().

First traceback:
  19: parallel:::.slaveRSOCK()
  18: slaveLoop(makeSOCKmaster(master, port, timeout, useXDR))
...

## End(Not run)

## bplasterror() reports the number of successful results and error message.
## Not run:
> bplasterror()
2 / 3 partial results stored. First 1 error messages:
[2]: Error in FUN(...): non-numeric argument to mathematical function

## End(Not run)

## ------------------------------------------------------------------------
## Resume calculations:
## ------------------------------------------------------------------------

## The resume mechanism attempts to re-run the data element that failed.
## In our example the character "two" list element will never succeed. In
## the runs below we replace the character with a numeric and compute the
## result.

## There are two methods for resuming a suspended calculation, one
## approach is to wrap the original call in bpresume().
## Not run:
> bpresume(res <- bplapply(list(1, 2, 3), sqrt, BPPARAM = snowp))
Resuming previous calculation...
> res
[[1]]
[1] 1

[[2]]
[1] 1.414214
```

```
[[3]]
[1] 1.732051

## End(Not run)

## An equivalent approach is to set BPRESUME = TRUE in bplapply().
## Not run:
res <- bplapply(list(1, 2, 3), sqrt, BPPARAM = snowp, BPRESUME = TRUE)

## End(Not run)
```

---

bpschedule *Schedule back-end Params*

Description

Use functions on this page to influence scheduling of parallel processing.

Usage

```
bpschedule(x, ...)
```

Arguments

x	An instance of a BiocParallelParam class, e.g., MulticoreParam, SnowParam, DoparParam. x can be missing, in which case the default back-end (see register) is used.
...	Additional arguments, perhaps used by methods.

Details

bpschedule returns a logical(1) indicating whether the parallel evaluation should occur at this point.

Value

bpschedule returns a scalar logical.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org.

See Also

BiocParallelParam for possible values of x.

Examples

```
bpschedule(SnowParam())  # TRUE
bpschedule(MulticoreParam(2)) # FALSE on windows

p <- MulticoreParam(recursive=FALSE)
bpschedule(p)                          # TRUE
bplapply(1:2, function(i, p) {
    bpschedule(p)                      # FALSE
}, p = p, BPPARAM=p)
```

---

bpvec *Parallel, vectorized evaluation*

Description

bpvec applies FUN to subsets of X. Any type of object X is allowed, provided length, [, and c methods are available. The return value is a vector of length equal to X, as with FUN(X).

Usage

```
bpvec(X, FUN, ..., AGGREGATE=c, BPPARAM=bpparam())

## S4 method for signature ANY,ANY
bpvec(X, FUN, ..., AGGREGATE=c, BPPARAM=bpparam())

## S4 method for signature ANY,BiocParallelParam
bpvec(X, FUN, ..., AGGREGATE=c, BPPARAM=bpparam())

## S4 method for signature ANY,missing
bpvec(X, FUN, ..., AGGREGATE=c, BPPARAM=bpparam())
```

Arguments

X	Any object for which methods length, [, and c are implemented.
FUN	The function to be applied to subsets of X.
...	Additional arguments for FUN.
AGGREGATE	A function taking any number of arguments ... called to reduce results (elements of the ... argument of AGGREGATE from parallel jobs. The default, c, concatenates objects and is appropriate for vectors; rbind might be appropriate for data frames.
BPPARAM	A optional BiocParallelParam instance determining the parallel back-end to be used during evaluation.

Details

When BPPARAM is an instance of a class derived from BiocParallelParam for which no other method applies, this method creates a vector of indexes and uses these in conjunction with bplapply to arrange for parallel evaluation.

When BPPARAM is a class for which no method is defined (e.g., SerialParam), FUN(X) is used.

See showMethods{bpvec} for additional methods, e.g., method?bpvec("MulticoreParam").

Value

The result should be identical to FUN(X, ...) (assuming that AGGREGATE is set appropriately).

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org. Original code as attributed in pvec.

See Also

bplapply for parallel lapply.

BiocParallelParam for possible values of BPPARAM.

Examples

```
showMethods("bpvec")

## ten tasks (1:10), called with as many back-end elements are specified
## by BPPARAM.  Compare with bplapply
system.time(result <- bpvec(1:10, function(v) {
    message("working") ## 10 tasks
    sqrt(v)
}))
result
```

---

bpvectorize *Transform vectorized functions into parallelized, vectorized function*

Description

This transforms a vectorized function into a parallel, vectorized function. Any function FUN can be used, provided its parallelized argument (by default, the first argument) has a length and [method defined, and the return value of FUN can be concatenated with c.

Usage

```
bpvectorize(FUN, ..., BPPARAM=bpparam())

## S4 method for signature ANY,ANY
bpvectorize(FUN, ..., BPPARAM=bpparam())

## S4 method for signature ANY,missing
bpvectorize(FUN, ..., BPPARAM=bpparam())
```

Arguments

FUN	A function whose first argument has a length and can be subset [, and whose evaluation would benefit by splitting the argument into subsets, each one of which is independently transformed by FUN. The return value of FUN must support concatenation with c.
...	Additional arguments to parallization, unused.
BPPARAM	An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation.

Details

The result of bpvectorize is a function with signature ...; arguments to the returned function are the original arguments FUN. BPPARAM is used for parallel evaluation.

When BPPARAM is a class for which no method is defined (e.g., SerialParam), FUN(X) is used.

See showMethods{bpvectorize} for additional methods, if any.

Value

A function taking the same arguments as FUN, but evaluated using bpvec for parallel evaluation across available cores.

Author(s)

Ryan Thompson mailto:rct@thompsonclan.org

See Also

bpvec

Examples

```
psqrt <- bpvectorize(sqrt) ## default parallelization
psqrt(1:10)
```

DoparParam-class *Enable parallel evaluation using registered dopar backend*

Description

This class is used to dispatch parallel operations to the dopar backend registered with the foreach package.

Usage

```
DoparParam(catch.errors = TRUE)
```

Arguments

catch.errors	logical(1)Flag to determine in apply-like functions (see e.g. bplapply) whether to quit with an error as soon as one application fails or encapsulation of function calls in try blocks which triggers a resume mechanism (see bpresume). Defaults to TRUE.

DoparParam constructor

Return a proxy object that dispatches parallel evaluation to the registered foreach parallel backend.

There are no options to the constructor. All configuration should be done through the normal interface to the foreach parallel backends.

Methods

The following generics are implemented and perform as documented on the corresponding help page (e.g., ?bpisup): bpworkers, bpstart, bpstop, bpisup, bpbackend, bpbackend<-, bpvec.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org

See Also

getClass("BiocParallelParam") for additional parameter classes.

register for registering parameter classes for use in parallel evaluation.

foreach-package for the parallel backend infrastructure used by this param class.

Examples

```
# First register a parallel backend with foreach
library(doParallel)
cl <- makeCluster(2)
registerDoParallel(cl)
```

```
p <- DoparParam()
bplapply(1:10, sqrt, BPPARAM=p)
bpvec(1:10, sqrt, BPPARAM=p)

stopCluster(cl)
## Not run:
register(DoparParam(), default=TRUE)

## End(Not run)
```

`MulticoreParam-class` *Enable multi-core parallel evaluation*

Description

This class is used to parameterize single computer multicore parallel evaluation on non-Windows computers. `multicoreWorkers()` chooses the number of workers based on operating system (Windows only supports 1 core), global user preference (`options(mc.cores=...)`), or the minimum of 8 and the number of detected cores (`detectCores()`).

Usage

```
MulticoreParam(workers = multicoreWorkers(), catch.errors = TRUE,
    setSeed = TRUE, recursive = TRUE, cleanup = TRUE,
    cleanupSignal = tools::SIGTERM, verbose = FALSE, ...)
multicoreWorkers()
```

Arguments

`workers`	`integer(1)`Number of workers on with Multicore and SSH backend which defaults here to all workers available. On managed HPC `workers` defaults to `NA` but can be set to control chunking of jobs. See argument `n.chunks` in `chunk` and `submitJobs` for more information.
`catch.errors`	`logical(1)`Flag to determine in apply-like functions (see e.g. `bplapply`) whether to quit with an error as soon as one application fails or encapsulation of function calls in `try` blocks which triggers a resume mechanism (see `bpresume`). Defaults to `TRUE`.
`setSeed`	`logical(1)`, as described in `parallel::mcparallel` argument `mc.set.seed`.
`recursive`	`logical(1)` indicating whether recursive calls are evaluated in parallel; see `parallel::mclapply` argument `mc.allow.recursive`.
`cleanup`	`logical(1)` indicating whether forked children will be terminated before `bplapply` returns, as for `parallel::mclapply` argument `cleanup`. If `TRUE`, then the signal sent to the child is `cleanupSignal`.
`cleanupSignal`	`integer(1)` the signal sent to forked processes when `cleanup=TRUE`.
`verbose`	`logical(1)` when `TRUE` echo stdout of forked processes. This is the complement of `parallel::mclapply`'s argument `mc.silent`.
`...`	Additional arguments passed to `mclapply`

MulticoreParam constructor

Return an object with specified values. The object may be saved to disk or reused within a session.

Methods

The following generics are implemented and perform as documented on the corresponding help page (e.g., ?bpisup): bpworkers, bpstart, bpstop, bpisup, bpschedule, bpbackend.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org

See Also

getClass("BiocParallelParam") for additional parameter classes.

register for registering parameter classes for use in parallel evaluation.

Examples

```
multicoreWorkers()
p <- MulticoreParam()
bplapply(1:10, sqrt, BPPARAM=p)
bpvec(1:10, sqrt, BPPARAM=p)

## Not run:
register(MulticoreParam(), default=TRUE)

## End(Not run)
```

---

register *Maintain a global registry of available back-end Params*

Description

Use functions on this page to add to or query a registry of back-ends, including the default for use when no BPPARAM object is provided to functions.

Usage

```
register(BPPARAM, default=TRUE)
registered(bpparamClass)
bpparam(bpparamClass)
```

Arguments

BPPARAM	An instance of a BiocParallelParam class, e.g., MulticoreParam, SnowParam, DoparParam.
default	Make this the default BiocParallelParam for subsequent evaluations? If FALSE, the argument is placed at the lowest priority position.
bpparamClass	When present, the text name of the BiocParallelParam class (e.g., "MulticoreParam") to be retrieved from the registry. When absent, a list of all registered instances is returned.

Details

The registry is a list of back-ends with configuration parameters for parallel evaluation. The first list entry is the default and is used by BiocParallel functions when no BPPARAM argument is supplied.

By default, the registry includes MulticoreParam, SnowParam, BatchJobsParam and SerialParam. The objects are constructed from global options of the corresponding name, or from the default constructor (e.g., SnowParam()) if no option is specified. The user can set customizations during start-up (e.g., in an .Rprofile file) with, for instance, options(MulticoreParam=quote(MulticoreParam(workers=8))).

The act of "registering" a back-end modifies the existing BiocParallelParam in the list; only one param of each type can be present in the registry. When default=TRUE, the newly registered param is moved to the top of the list thereby making it the default. When default=FALSE, the param is modified 'in place' vs being moved to the top.

bpparam(), invoked with no arguments, returns the default BiocParallelParam instance from the registry. When called with the text name of a bpparamClass, the global options are consulted first, e.g., options(MulticoreParam=MulticoreParam()) and then the value of registered(bpparamClass).

Value

register returns, invisibly, a list of registered back-ends.

registered returns the back-end of type bpparamClass or, if bpparamClass is missing, a list of all registered back-ends.

bpparam returns the back-end of type bpparamClass or,

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org.

See Also

BiocParallelParam for possible values of BPPARAM.

Examples

```
## ------------------------------------------------------------------------
## The registry
## ------------------------------------------------------------------------
```

```
## The full registry.
registered()

## When default = TRUE the last param registered becomes the new default.
snowparam <- SnowParam(workers = 3, type = "PSOCK")
register(snowparam, default = TRUE)
registered()

## Retrieve the default back-end,
bpparam()

## or a specific BiocParallelParam.
bpparam("MulticoreParam")

## ------------------------------------------------------------------------
## Specifying a back-end for evaluation
## ------------------------------------------------------------------------

## The back-end of choice is specified in the BPPARAM argument to
## the BiocParallel functions. None, one, or multiple back-ends can be
## provided.

## BPPARAM not specified:
## The default back-end from the registry is used.
bplapply(1:6, sqrt)

## BPPARAM as a single param:
bplapply(1:6, sqrt, BPPARAM = MulticoreParam(3))

## BPPARAM as a list of params:
## Useful for multiple levels of parallel evaluation. The 1:6 are divided
## among Snow workers, each of which dispatch to Multicore workers.
myfun <- function(x, BPPARAM, ...) {
  library(BiocParallel)
  res <- bplapply(x, sqrt, BPPARAM = BPPARAM)
  do.call(mean, res)
}
params <- list(bpparam("SnowParam"), bpparam("MulticoreParam"))
bplapply(list(1:2, 3:4, 5:6), myfun, BPPARAM = params)
```

`SerialParam-class` *Enable serial evaluation*

Description

This class is used to parameterize serial evaluation, primarily to facilitate easy transition from parallel to serial code.

Usage

```
SerialParam(catch.errors = TRUE)
```

Arguments

catch.errors	logical(1)Flag to determine in apply-like functions (see e.g. bplapply) whether to quit with an error as soon as one application fails or encapsulation of function calls in try blocks which triggers a resume mechanism (see bpresume). Defaults to TRUE.

SerialParam constructor

Return an object to be used for serial evaluation of otherwise parallel functions such as bplapply, bpvec.

Methods

The following generics are implemented and perform as documented on the corresponding help page (e.g., ?bpworkers): bpworkers. bpisup, bpstart, bpstop, are implemented, but do not have any side-effects.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org

See Also

getClass("BiocParallelParam") for additional parameter classes.

register for registering parameter classes for use in parallel evaluation.

Examples

```
p <- SerialParam()
simplify2array(bplapply(1:10, sqrt, BPPARAM=p))
bpvec(1:10, sqrt, BPPARAM=p)

## Not run:
register(SerialParam(), default=TRUE)

## End(Not run)
```

SnowParam-class *Enable simple network of workstations (SNOW)-style parallel evaluation*

Description

This class is used to parameterize simple network of workstations (SNOW) parallel evaluation on one or several physical computers. snowWorkers() chooses the number of workers based on global user preference (options(mc.cores=...)), or the minimum of 8 and the number of dectected cores (detectCores()).

Usage

```
SnowParam(workers = snowWorkers(), type=c("SOCK", "PSOCK", "FORK", "MPI"),
    catch.errors = TRUE, ...)

## invoke as(cl, "SnowParam")
## S4 method for signature SOCKcluster,SnowParam
coerce(from, to)
## S4 method for signature spawnedMPIcluster,SnowParam
coerce(from, to)

snowWorkers()
```

Arguments

workers	integer(1)Number of workers on with Multicore and SSH backend which defaults here to all workers available. On managed HPC workers defaults to NA but can be set to control chunking of jobs. See argument n.chunks in chunk and submitJobs for more information.
type	character(1) type of cluster to use, as described in clusterMap argument type. Use MulticoreParam instead of type=FORK.
catch.errors	logical(1) Flag to determine in apply-like functions (see e.g. bplapply) whether to quit with an error as soon as one application fails or encapsulation of function calls in try blocks which triggers a resume mechanism (see bpresume). Defaults to TRUE.
...	Additional arguments passed to makeCluster
from, to	(N.B. Use as(from, "SnowParam") to coerce from a cluster created with, e.g., parallel::makeCluster). from is a SOCKcluster or derived instance (e.g., from parallel::makeCluster), to be coerced to a SnowParam instance.

SnowParam constructor

Return an object representing a SNOW cluster. The cluster is not created until bpstart is called.

bpstart creates the cluster by invoking makeCluster with arguments spec=workers, type, and other arguments passed to ... in SnowParam.

Use as(cl, "SnowParam") to coerce a cluster created directly by parallel::param to a SnowParam instance. Instances created in this way cannot be started or stopped.

Methods

The following generics are implemented and perform as documented on the corresponding help page (e.g., ?bpisup): bpworkers, bpstart, bpstop, bpisup, bpbackend, bpbackend<-, bpvec.

Author(s)

Martin Morgan mailto:mtmorgan@fhcrc.org

See Also

`getClass("BiocParallelParam")` for additional parameter classes.

`register` for registering parameter classes for use in parallel evaluation.

Examples

```
snowWorkers()
p <- SnowParam(2L)
bplapply(1:10, sqrt, BPPARAM=p)
bpvec(1:10, sqrt, BPPARAM=p)

## Not run:
register(SnowParam(2L), default=TRUE)

## End(Not run)
```

Index