

# Package ‘pbdMPI’

April 5, 2018

**Version** 0.3-5

**Date** 2018-04-04

**Title** Programming with Big Data -- Interface to MPI

**Depends** R (>= 3.3.0), methods, rlecuyer

**Enhances** pbdPROF, pbdZMQ

**LazyLoad** yes

**LazyData** yes

**Description** An efficient interface to MPI by utilizing S4 classes and methods with a focus on Single Program/Multiple Data ('SPMD') parallel programming style, which is intended for batch parallel execution.

**SystemRequirements** OpenMPI (>= 1.5.4) on Solaris, Linux, Mac, and FreeBSD. MS-MPI (Microsoft MPI v7.1 (SDK) and Microsoft HPC Pack 2012 R2 MS-MPI Redistributable Package) on Windows.

**License** Mozilla Public License 2.0

**URL** <http://r-pbd.org/>

**BugReports** <https://github.com/snoweye/pbdMPI/issues>

**MailingList** Please send questions and comments regarding pbdR to RBigData@gmail.com

**NeedsCompilation** yes

**Maintainer** Wei-Chen Chen <wccsnow@gmail.com>

**Author** Wei-Chen Chen [aut, cre],  
George Ostrouchov [aut],  
Drew Schmidt [aut],  
Pragneshkumar Patel [aut],  
Hao Yu [aut],  
Christian Heckendorf [ctb] (FreeBSD),  
Brian Ripley [ctb] (Windows HPC Pack 2012),  
R Core team [ctb] (some functions are modified from the base packages)

**Repository** CRAN

**Date/Publication** 2018-04-05 05:49:41 UTC

**R topics documented:**

pbdMPI-package	3
allgather-method	5
allreduce-method	7
alltoall	8
apply and lapply	10
bcast-method	13
communicator	14
gather-method	17
get job id	19
global all pairs	21
global any and all	22
global as.gbd	24
global balanc	26
global base	27
global distance function	29
global match.arg	31
global pairwise	32
global print and cat	34
global range, max, and min	36
global reading	37
global Rprof	40
global sort	41
global stop and warning	42
global timer	44
global which, which.max, and which.min	45
global writing	47
info	48
irecv-method	50
is.comm.null	51
isend-method	53
MPI array pointers	54
probe	55
recv-method	56
reduce-method	58
scatter-method	60
seed for RNG	62
send-method	64
sendrecv-method	65
sendrecv.replace-method	67
Set global pbd options	69
sourcetag	71
SPMD Control	72
SPMD Control Functions	74
Task Pull	74
Utility execmpi	76
wait	77

---

pbdMPI-package                      *Programming with Big Data – Interface to MPI*

---

## Description

pbdMPI provides an efficient interface to MPI by utilizing S4 classes and methods with a focus on Single Program/Multiple Data (SPMD) parallel programming style, which is intended for batch parallel execution.

## Details

```
Package:    pbdMPI
Type:      Package
License:   Mozilla Public License 2.0
LazyLoad:  yes
```

This package requires an MPI library (OpenMPI, MPICH2, or LAM/MPI). The install command (with OpenMPI library) is

```
> tar zxvf pbdMPI_0.1-0.tar.gz
> R CMD INSTALL pbdMPI
```

Other arguments include

Argument	Default
<code>--with-mpi-type</code>	OPENMPI
<code>--with-mpi-include</code>	<code>\${MPI_ROOT}/include</code>
<code>--with-mpi-libpath</code>	<code>\${MPI_ROOT}/lib</code>
<code>--with-mpi</code>	<code>\${MPI_ROOT}</code>

where `${MPI_ROOT}` is the path to the MPI root. See the package source file `pbdMPI/configure` for details.

After loading `library(pbdMPI)`, the standard process starts from `init()` which set two global variables `.comm.size` and `.comm.rank`. The standard process should end with `finalize()`.

Most functions are assumed to run in SPMD, i.e. in batch mode. Ideally, most codes run with `mpiexec` and `Rscript`, together, such as

```
> mpiexec -np 2 Rscript some_code.r
```

where `some_code.r` contains whole SPMD program.

The package source files provide several examples based on **pbdMPI**, such as

Directory	Examples
pbdMPI/inst/examples/test_spm/	major SPMD functions
pbdMPI/inst/examples/test_rmpi/	analog to <b>Rmpi</b>
pbdMPI/inst/examples/test_parallel/	analog to <b>parallel</b>
pbdMPI/inst/examples/test_performance/	performance tests
pbdMPI/inst/examples/test_s4/	S4 extension
pbdMPI/inst/examples/test_cs/	client/server examples
pbdMPI/inst/examples/test_long_vector/	long vector examples

where test\_long\_vector needs to recompile with setting

```
#define MPI_LONG_DEBUG 1
in pbdMPI/src/pkg_constant.h.
```

The current version is mainly written and tested under OpenMPI environments in Linux system (xubuntu-11.04). Also, it is tested under MPICH2 environments in Windows 7 system. It is expected to be fine for other MPI libraries and other OS platforms.

### Author(s)

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### See Also

[allgather\(\)](#), [allreduce\(\)](#), [bcast\(\)](#), [gather\(\)](#), [reduce\(\)](#), [scatter\(\)](#).

### Examples

```
## Not run:
### Under command mode, run the demo with 2 processors by
### (Use Rscript.exe for windows system)
# mpiexec -np 2 Rscript -e "demo(allgather,'pbdMPI',ask=F,echo=F)"
# mpiexec -np 2 Rscript -e "demo(allreduce,'pbdMPI',ask=F,echo=F)"
# mpiexec -np 2 Rscript -e "demo(bcast,'pbdMPI',ask=F,echo=F)"
# mpiexec -np 2 Rscript -e "demo(gather,'pbdMPI',ask=F,echo=F)"
# mpiexec -np 2 Rscript -e "demo(reduce,'pbdMPI',ask=F,echo=F)"
# mpiexec -np 2 Rscript -e "demo(scatter,'pbdMPI',ask=F,echo=F)"
### Or
# execmpi("demo(allgather,'pbdMPI',ask=F,echo=F)", nranks = 2L)
# execmpi("demo(allreduce,'pbdMPI',ask=F,echo=F)", nranks = 2L)
# execmpi("demo(bcast,'pbdMPI',ask=F,echo=F)", nranks = 2L)
# execmpi("demo(gather,'pbdMPI',ask=F,echo=F)", nranks = 2L)
# execmpi("demo(reduce,'pbdMPI',ask=F,echo=F)", nranks = 2L)
# execmpi("demo(scatter,'pbdMPI',ask=F,echo=F)", nranks = 2L)

## End(Not run)
```

---

allgather-method      *All Ranks Gather Objects from Every Rank*

---

### Description

This method lets all ranks gather objects from every rank in the same communicator. The default return is a list of length equal to `comm.size(comm)`.

### Usage

```
allgather(x, x.buffer = NULL, x.count = NULL, displs = NULL,
          comm = .pbd_env$SPMD.CT$comm,
          unlist = .pbd_env$SPMD.CT$unlist)
```

### Arguments

<code>x</code>	an object to be gathered from all ranks.
<code>x.buffer</code>	a buffer to hold the return object which probably has 'size of <code>x</code> ' times ' <code>comm.size(comm)</code> ' with the same type of <code>x</code> .
<code>x.count</code>	a vector of length ' <code>comm.size</code> ' containing all object lengths.
<code>displs</code>	<code>c(0L, cumsum(x.count))</code> by default.
<code>comm</code>	a communicator number.
<code>unlist</code>	if unlist the return.

### Details

All `x` on all ranks are likely presumed to have the same size and type.

`x.buffer`, `x.count`, and `displs` can be `NULL` or unspecified. If specified, the type should be one of integer, double, or raw specified correctly according to the type of `x`.

If `x.count` is specified, then the `spmd.allgather.v.*()` is called.

### Value

A list of length `comm.size(comm)` is returned by default.

### Methods

For calling `spmd.allgather.*()`:

```
signature(x = "ANY", x.buffer = "missing", x.count = "missing")
```

```
signature(x = "integer", x.buffer = "integer", x.count = "missing")
```

```
signature(x = "numeric", x.buffer = "numeric", x.count = "missing")
```

```
signature(x = "raw", x.buffer = "raw", x.count = "missing")
```

For calling `spmd.allgather.v.*`:

```
signature(x = "ANY", x.buffer = "missing", x.count = "integer")
signature(x = "ANY", x.buffer = "ANY", x.count = "integer")
signature(x = "integer", x.buffer = "integer", x.count = "integer")
signature(x = "numeric", x.buffer = "numeric", x.count = "integer")
signature(x = "raw", x.buffer = "raw", x.count = "integer")
```

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### See Also

[gather\(\)](#), [allreduce\(\)](#), [reduce\(\)](#).

### Examples

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.file <- tempfile()
cat("
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
y <- allgather(matrix(x, nrow = 1))
comm.print(y)

y <- allgather(x, double(N * .comm.size))
comm.print(y)

### Finish.
finalize()
", file = spmd.file)
pbdMPI::execmpi(spmd.file = spmd.file, nrank = 2L)
```

---

allreduce-method      *All Ranks Receive a Reduction of Objects from Every Rank*

---

### Description

This method lets all ranks receive a deduction of objects from every rank in the same communicator based on a given operation. The default return is an object like the input.

### Usage

```
allreduce(x, x.buffer = NULL, op = .pbd_env$SPMD.CT$op,  
          comm = .pbd_env$SPMD.CT$comm)
```

### Arguments

x	an object to be gathered from all ranks.
x.buffer	a buffer to hold the return object which probably has x with the same type of x.
op	a reduction operation applied to combine all x.
comm	a communicator number.

### Details

All x on all ranks are likely presumed to have the same size and type.

x.buffer can be NULL or unspecified. If specified, the type should be one of integer, double, or raw specified correctly according to the type of x.

### Value

The reduced object of the same type as x is returned by default.

### Methods

For calling `spmd.allreduce.*`:

```
signature(x = "ANY", x.buffer = "missing")  
signature(x = "integer", x.buffer = "integer")  
signature(x = "numeric", x.buffer = "numeric")  
signature(x = "logical", x.buffer = "logical")  
signature(x = "float32", x.buffer = "float32")
```

### Author(s)

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## See Also

[allgather\(\)](#), [gather\(\)](#), [reduce\(\)](#).

## Examples

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

smpd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
y <- allreduce(matrix(x, nrow = 1), op = "sum")
comm.print(y)

y <- allreduce(x, double(N), op = "prod")
comm.print(y)

comm.set.seed(1234, diff = TRUE)
x <- as.logical(round(runif(N)))
y <- allreduce(x, logical(N), op = "land")
comm.print(y)

### Finish.
finalize()
"
pbdMPI::execmpi(smpd.code = smpd.code, nranks = 2L)
```

---

alltoall

*All to All*

---

## Description

These functions make calls to `MPI_Alltoall()` and `MPI_Alltoallv()`.



**Usage**

```
spmd.alltoall.integer(x.send, x.recv, send.count, recv.count,  
                      comm = .pbd_env$SPMD.CT$comm)  
spmd.alltoall.double(x.send, x.recv, send.count, recv.count,  
                     comm = .pbd_env$SPMD.CT$comm)  
spmd.alltoall.raw(x.send, x.recv, send.count, recv.count,  
                  comm = .pbd_env$SPMD.CT$comm)  
  
spmd.alltoallv.integer(x.send, x.recv, send.count, recv.count,  
                       sdispls, rdispls, comm = .pbd_env$SPMD.CT$comm)  
spmd.alltoallv.double(x.send, x.recv, send.count, recv.count,  
                      sdispls, rdispls, comm = .pbd_env$SPMD.CT$comm)  
spmd.alltoallv.raw(x.send, x.recv, send.count, recv.count,  
                   sdispls, rdispls, comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

x.send	an object to send.
x.recv	an object to receive
send.count	send counter
recv.count	recv counter
sdispls	send dis pls
rdispls	recv dis pls
comm	a communicator number.

**Details**

These are very low level functions. Use with cautions. Neither S4 method nor long vector is supported.

**Value**

These are very low level functions. Use with cautions. Neither S4 method nor long vector is supported.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[allgather\(\)](#), [allgatherv\(\)](#).

**Examples**

```

## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript --vanilla [...].r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
n <- as.integer(2)
x <- 1:(.comm.size * n)
comm.cat("\nOriginal x:\n", quiet = TRUE)
comm.print(x, all.rank = TRUE)

x <- as.integer(x)
y <- spmd.alltoall.integer(x, integer(length(x)), n, n)
comm.cat("\nAlltoall y:\n", quiet = TRUE)
comm.print(y, all.rank = TRUE)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)

```

---

apply and lapply

*Parallel Apply and Lapply Functions*

---

**Description**

The functions are parallel versions of apply and lapply functions.

**Usage**

```

pbdApply(X, MARGIN, FUN, ..., pbd.mode = c("mw", "spmd", "dist"),
         rank.source = .pbd_env$SPMD.CT$rank.root,
         comm = .pbd_env$SPMD.CT$comm,
         barrier = TRUE)
pbdLapply(X, FUN, ..., pbd.mode = c("mw", "spmd", "dist"),
         rank.source = .pbd_env$SPMD.CT$rank.root,
         comm = .pbd_env$SPMD.CT$comm,
         bcast = FALSE, barrier = TRUE)
pbdSapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE,
         pbd.mode = c("mw", "spmd", "dist"),

```

```
rank.source = .pbd_env$SPMD.CT$rank.root,
comm = .pbd_env$SPMD.CT$comm,
bcast = FALSE, barrier = TRUE)
```

### Arguments

X	a matrix or array in <code>pbdApply()</code> or a list in <code>pbdLapply()</code> and <code>pbdSapply()</code> .
MARGIN	MARGIN as in the <code>apply()</code> .
FUN	as in the <code>apply()</code> .
...	optional arguments to FUN.
simplify	as in the <code>sapply()</code> .
USE.NAMES	as in the <code>sapply()</code> .
pbd.mode	mode of distributed data X.
rank.source	a rank of source where X broadcast from.
comm	a communicator number.
bcast	if bcast to all ranks.
barrier	if barrier for all ranks.

### Details

All functions are majorly called in manager/workers mode (`pbd.model = "mw"`), and just work the same as their serial version.

If `pbd.mode = "mw"`, the X in `rank.source` (master) will be redistributed to processors (workers), then apply FUN on the new data, and results are gathered to `rank.source`. “In SPMD, master is one of workers.” ... is also `scatter()` from `rank.source`.

If `pbd.mode = "spmd"`, the same copy of X is supposed to exist in all processors, and original `apply()`, `lapply()`, or `sapply()` is operated on part of X. An `allgather()` or `gather()` call is required to aggregate results manually.

If `pbd.mode = "dist"`, the different X is supposed to exist in all processors, i.e. ‘distinct or distributed’ X, and original `apply()`, `lapply()`, or `sapply()` is operated on the all X. An `allgather()` or `gather()` call is required to aggregate results manually.

In SPMD, it is better to split data into pieces, and X is a local matrix in all processors. Originally, `apply()` should be sufficient in this case.

### Value

A list or matrix will be returned.

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```

### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Example for pbdApply.
N <- 100
x <- matrix((1:N) + N * .comm.rank, ncol = 10)
y <- pbdApply(x, 1, sum, pbd.mode = "mw")
comm.print(y)

y <- pbdApply(x, 1, sum, pbd.mode = "spmd")
comm.print(y)

y <- pbdApply(x, 1, sum, pbd.mode = "dist")
comm.print(y)

### Example for pbdApply for 3D array.
N <- 60
x <- array((1:N) + N * .comm.rank, c(3, 4, 5))
dimnames(x) <- list(lat = paste("lat", 1:3, sep = "\"),
                  lon = paste("lon", 1:4, sep = "\"),
                  time = paste("time", 1:5, sep = "\"))
comm.print(x[, , 1:2])

y <- pbdApply(x, c(1, 2), sum, pbd.mode = "mw")
comm.print(y)

y <- pbdApply(x, c(1, 2), sum, pbd.mode = "spmd")
comm.print(y)

y <- pbdApply(x, c(1, 2), sum, pbd.mode = "dist")
comm.print(y)

### Example for pbdLapply.
N <- 100
x <- split((1:N) + N * .comm.rank, rep(1:10, each = 10))
y <- pbdLapply(x, sum, pbd.mode = "mw")
comm.print(unlist(y))

y <- pbdLapply(x, sum, pbd.mode = "spmd")
comm.print(unlist(y))

```

```
y <- pbdLapply(x, sum, pbd.mode = \"dist\")
comm.print(unlist(y))

### Finish.
finalize()
"
pbdMPI::execmpi(spmd.code, nranks = 2L)
```

---

bcast-method

*A Rank Broadcast an Object to Every Rank*

---

## Description

This method lets a rank broadcast an object to every rank in the same communicator. The default return is the object.

## Usage

```
bcast(x, rank.source = .pbd_env$SPMD.CT$rank.source,
      comm = .pbd_env$SPMD.CT$comm)
```

## Arguments

x	an object to be broadcast from all ranks.
rank.source	a rank of source where x broadcast from.
comm	a communicator number.

## Details

The same copy of x is sent to all ranks.

## Value

Every rank has x returned.

## Methods

For calling `spmd.bcast.*`:

```
signature(x = "ANY")
signature(x = "integer")
signature(x = "numeric")
signature(x = "raw")
```

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[scatter\(\)](#).

**Examples**

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
x <- matrix(1:5, nrow = 1)
y <- bcast(x)
comm.print(y)

### Finish.
finalize()
"
pbdMPI::execmpi(spmd.code, nrank = 2L)
```

---

communicator

*Communicator Functions*

---

**Description**

The functions provide controls to communicators.

**Usage**

```
barrier(comm = .pbd_env$SPMD.CT$comm)
comm.is.null(comm = .pbd_env$SPMD.CT$comm)
comm.rank(comm = .pbd_env$SPMD.CT$comm)
comm.localrank(comm = .pbd_env$SPMD.CT$comm)
comm.size(comm = .pbd_env$SPMD.CT$comm)
comm.dup(comm, newcomm)
```

```

comm.free(comm = .pbd_env$SPMD.CT$comm)
init(set.seed = TRUE)
finalize(mpi.finalize = .pbd_env$SPMD.CT$mpi.finalize)
is.finalized()

comm.abort(errorcode = 1, comm = .pbd_env$SPMD.CT$comm)
comm.split(comm = .pbd_env$SPMD.CT$comm, color = 0L, key = 0L,
           newcomm = .pbd_env$SPMD.CT$newcomm)
comm.disconnect(comm = .pbd_env$SPMD.CT$comm)
comm.connect(port.name, info = .pbd_env$SPMD.CT$info,
             rank.root = .pbd_env$SPMD.CT$rank.root,
             comm = .pbd_env$SPMD.CT$comm,
             newcomm = .pbd_env$SPMD.CT$newcomm)
comm.accept(port.name, info = .pbd_env$SPMD.CT$info,
            rank.root = .pbd_env$SPMD.CT$rank.root,
            comm = .pbd_env$SPMD.CT$comm,
            newcomm = .pbd_env$SPMD.CT$newcomm)

port.open(info = .pbd_env$SPMD.CT$info)
port.close(port.name)
serv.publish(port.name, serv.name = .pbd_env$SPMD.CT$serv.name,
             info = .pbd_env$SPMD.CT$info)
serv.unpublish(port.name, serv.name = .pbd_env$SPMD.CT$serv.name,
               info = .pbd_env$SPMD.CT$info)
serv.lookup(serv.name = .pbd_env$SPMD.CT$serv.name,
            info = .pbd_env$SPMD.CT$info)

intercomm.merge(intercomm = .pbd_env$SPMD.CT$intercomm,
                high = 0L, comm = .pbd_env$SPMD.CT$comm)
intercomm.create(local.comm = .pbd_env$SPMD.CT$comm,
                 local.leader = .pbd_env$SPMD.CT$rank.source,
                 peer.comm = .pbd_env$SPMD.CT$intercomm,
                 remote.leader = .pbd_env$SPMD.CT$rank.dest,
                 tag = .pbd_env$SPMD.CT$tag,
                 newintercomm = .pbd_env$SPMD.CT$newcomm)

comm.c2f(comm = .pbd_env$SPMD.CT$comm)

```

### Arguments

<code>comm</code>	a communicator number.
<code>mpi.finalize</code>	if MPI should be shutdown.
<code>set.seed</code>	if a random seed preset.
<code>port.name</code>	a port name with default maximum length 1024 characters for OpenMPI.
<code>info</code>	a info number.
<code>rank.root</code>	a root rank.
<code>newcomm</code>	a new communicator number.

<code>color</code>	control of subset assignment.
<code>key</code>	control of rank assignment.
<code>serv.name</code>	a service name.
<code>errorcode</code>	an error code to abort MPI.
<code>intercomm</code>	a intercommunicator number.
<code>high</code>	used to order the groups within comm.
<code>local.comm</code>	a local communicator number.
<code>local.leader</code>	the leader number of local communicator.
<code>peer.comm</code>	a peer communicator number.
<code>remote.leader</code>	the remote leader number of peer communicator.
<code>newintercomm</code>	a new intercommunicator number.
<code>tag</code>	a tag number.

### Details

Another functions are direct calls to MPI library.

`barrier()` blocks all processors until everyone call this.

`comm.is.null()` returns -1 if the array of communicators is not allocated, i.e. `init()` is not called yet. It returns 1 if the communicator is not initialized, i.e. NULL. It returns 0 if the communicator is initialized.

`comm.rank()` returns the processor's rank for the given comm.

`comm.size()` returns the total processes for the given comm.

`comm.dup()` duplicate a newcomm from comm.

`comm.free()` free a comm.

`init()` initializes a MPI world, and set two global variables `.comm.size` and `.comm.rank` in `.GlobalEnv`. A random seed will be preset by default (`Sys.getpid() + Sys.time()`) to the package **rlecuyer**.

`finalize()` frees memory and finishes a MPI world if `mpi.finalize = TRUE`. `is.finalized()` checks if MPI is already finalized.

`comm.abort()` aborts MPI.

`comm.split()` create a newcomm by color and key.

`comm.disconnect()` frees a comm.

`comm.connect()` connects a newcomm.

`comm.accept()` accepts a newcomm.

`port.open()` opens a port and returns the port name.

`port.close()` closes a port by name.

`serv.publish()` publishes a service via `port.name`.

`serv.unpublish()` unpublishes a service via `port.name`.

`serv.lookup()` lookup the `serv.name` and returns the port name.



intercomm.merge() merges the intercomm to intracommunicator.  
intercomm.create() creates a new intercomm from two peer intracommunicators.  
comm.c2f() returns an integer for Fortran MPI support.

### Value

Most function return an invisible state of MPI call.

### Author(s)

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### Examples

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples .
comm.print(.comm.size)
comm.print(.comm.rank, all.rank = TRUE)
comm.print(comm.rank(), rank.print = 1)
comm.print(comm.c2f())

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)
```

---

gather-method

*A Rank Gathers Objects from Every Rank*


---

### Description

This method lets a rank gather objects from every rank in the same communicator. The default return is a list of length equal to 'comm size'.

**Usage**

```
gather(x, x.buffer = NULL, x.count = NULL, displs = NULL,
       rank.dest = .pbd_env$SPMD.CT$rank.root,
       comm = .pbd_env$SPMD.CT$comm,
       unlist = .pbd_env$SPMD.CT$unlist)
```

**Arguments**

<code>x</code>	an object to be gathered from all ranks.
<code>x.buffer</code>	a buffer to hold the return object which probably has 'size of x' times 'comm size' with the same type of x.
<code>x.count</code>	a vector of length 'comm size' containing all object lengths.
<code>displs</code>	<code>c(0L, cumsum(x.count))</code> by default.
<code>rank.dest</code>	a rank of destination where all x gather to.
<code>comm</code>	a communicator number.
<code>unlist</code>	if unlist the return.

**Details**

All x on all ranks are likely presumed to have the same size and type.

`x.buffer`, `x.count`, and `displs` can be NULL or unspecified. If specified, the type should be one of integer, double, or raw specified correctly according to the type of x.

If `x.count` is specified, then the `spmd.gatherv.*()` is called.

**Value**

If `rank.dest == comm.rank(comm)`, then a list of length 'comm size' is returned by default. Otherwise, NULL is returned.

**Methods**

For calling `spmd.gather.*()`:

```
signature(x = "ANY", x.buffer = "missing", x.count = "missing")
signature(x = "integer", x.buffer = "integer", x.count = "missing")
signature(x = "numeric", x.buffer = "numeric", x.count = "missing")
signature(x = "raw", x.buffer = "raw", x.count = "missing")
```

For calling `spmd.gatherv.*()`:

```
signature(x = "ANY", x.buffer = "missing", x.count = "integer")
signature(x = "ANY", x.buffer = "ANY", x.count = "integer")
signature(x = "integer", x.buffer = "integer", x.count = "integer")
signature(x = "numeric", x.buffer = "numeric", x.count = "integer")
signature(x = "raw", x.buffer = "raw", x.count = "integer")
```

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[gather\(\)](#), [allreduce\(\)](#), [reduce\(\)](#).

**Examples**

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

smpd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
y <- gather(matrix(x, nrow = 1))
comm.print(y)
y <- gather(x, double(N * .comm.size))
comm.print(y)

### Finish.
finalize()
"

pbdMPI::execmpi(smpd.code, nranks = 2L)
```

---

get job id

*Divide Job ID by Ranks*

---

**Description**

This function obtains job id which can be used to divide jobs.

**Usage**

```
get.jid(n, method = .pbd_env$SPMD.CT$divide.method[1], all = FALSE,
        comm = .pbd_env$SPMD.CT$comm, reduced = FALSE)
```

**Arguments**

n	total number of jobs.
method	a way to divide jobs.
all	indicate if return all id for each processor.
comm	a communicator number.
reduced	indicate if return should be a reduced representation.

**Details**

n is total number of jobs needed to be divided into all processors (`comm.size(comm)`), i.e. 1:n will be split according to the rank of processor (`comm.rank(comm)`) and method. Job id will be returned. Currently, three possible methods are provided.

"block" will use return id's which are nearly equal size blocks. For example, 7 jobs in 4 processors will have `jid=1` for rank 0, `jid=2,3` for rank 1, `jid=4,5` for rank 2, and `jid=6,7` for rank 3.

"block0" will use return id's which are nearly equal size blocks, in the opposite direction of "block". For example, 7 jobs in 4 processors will have `jid=1,2` for rank 0, `jid=3,4` for rank 1, `jid=5,6` for rank 2, and `jid=7` for rank 3.

"cycle" will use return id's which are nearly equal size in cycle. For example, 7 jobs in 4 processors will have `jid=1,5` for rank 0, `jid=2,6` for rank 1, `jid=3,7` for rank 2, and `jid=4` for rank 3.

**Value**

`get.id()` returns a vector containing job id for each individual processor if `all = FALSE`. While it returns a list containing all job id for all processor if `all = TRUE`. The list has length equal to `COMM.SIZE`.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[task.pull\(\)](#).

**Examples**

```
### Save code in a file "demo.r" and run with 4 processors by
### SHELL> mpiexec -np 4 Rscript demo.r

smd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
```

```

init()

### Examples.
comm.cat(">>> block\n", quiet = TRUE)
jid <- get.jid(7, method = "block")
comm.print(jid, all.rank = TRUE)

comm.cat(">>> cycle\n", quiet = TRUE)
jid <- get.jid(7, method = "cycle")
comm.print(jid, all.rank = TRUE)

comm.cat(">>> block (all)\n", quiet = TRUE)
alljid <- get.jid(7, method = "block", all = TRUE)
comm.print(alljid)

comm.cat(">>> cycle (all)\n", quiet = TRUE)
alljid <- get.jid(7, method = "cycle", all = TRUE)
comm.print(alljid)

### Finish.
finalize()
"
pbdMPI::execmpi(spmd.code, nranks = 4L)

```

---

global all pairs	<i>Global All Pairs</i>
------------------	-------------------------

---

## Description

This function provide global all pairs.

## Usage

```

comm.allpairs(N, diag = FALSE, symmetric = TRUE,
             comm = .pbd_env$SPMD.CT$comm)

```

## Arguments

N	number of elements for matching, (i, j) for all $1 \leq i, j \leq N$ .
diag	if matching the same elements, (i, i) for all i.
symmetric	if matching upper triangular elements. TRUE for $i \geq j$ only, otherwise for all (i, j).
comm	a communicator number.

## Details

The function generates all combinations of N elements.

**Value**

The function returns a gbd matrix in row blocks with 2 columns named i and j. The number of rows is dependent on the options `diag` and `symmetric`. If `diag = TRUE` and `symmetric = FALSE`, then this case has the maximum number of rows,  $N^2$ .

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[comm.dist\(\)](#).

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
id.matrix <- comm.allpairs(comm.size() + 1)
comm.print(id.matrix, all.rank = TRUE)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)
```

---

global any and all      *Global Any and All Functions*

---

**Description**

These functions are global any and all applying on distributed data for all ranks.

**Usage**

```
comm.any(x, na.rm = FALSE, comm = .pbd_env$SPMD.CT$comm)
comm.all(x, na.rm = FALSE, comm = .pbd_env$SPMD.CT$comm)

comm.allcommon(x, comm = .pbd_env$SPMD.CT$comm,
               lazy.check = .pbd_env$SPMD.CT$lazy.check)
```

**Arguments**

<code>x</code>	a vector.
<code>na.rm</code>	if NA removed or not.
<code>comm</code>	a communicator number.
<code>lazy.check</code>	if TRUE, then <code>allreduce</code> is used to check all ranks, otherwise, <code>allgather</code> is used.

**Details**

These functions will apply `any()` and `all()` locally, and apply `allgather()` to get all local results from other ranks, then apply `any()` and `all()` on all local results.

`comm.allcommon()` is to check if `x` is exactly the same across all ranks. This is a vectorized operation on `x` where the input and output have the same length of vector, while `comm.any()` and `comm.all()` return a scalar.

Note that `lazy.check = TRUE` is faster as number of cores is large, but it may cause some inconsistency in some cases. `lazy.check = FALSE` is much slower, but it provides more accurate checking.

**Value**

The global check values (TRUE, FALSE, NA) are returned to all ranks.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
```

```

init()

### Examples.
if(comm.rank() == 0){
  a <- c(T, F, NA)
} else{
  a <- T
}

comm.any(a)
comm.all(a)
comm.any(a, na.rm = TRUE)
comm.all(a, na.rm = TRUE)

comm.allcommon(1:3)
if(comm.rank() == 0){
  a <- 1:3
} else{
  a <- 3:1
}
comm.allcommon.integer(a)

### Finish.
finalize()
"
# execmpi(spmc.code, nranks = 2L)

## End(Not run)

```

---

global as.gbd

*Global As GBD Function*


---

## Description

This function redistributes a regular matrix existed in rank.source and turns it in a gbd matrix in row blocks.

## Usage

```

comm.as.gbd(X, balance.method = .pbd_env$SPMD.IO$balance.method,
            rank.source = .pbd_env$SPMD.CT$rank.source,
            comm = .pbd_env$SPMD.CT$comm)

```

## Arguments

X	a regular matrix in rank.source and to be redistributed as a gbd.
balance.method	a balance method.
rank.source	a rank of source where elements of x scatter from.
comm	a communicator number.



## Details

X matrix in rank.source will be redistributed as a gbd matrix in row blocks.

This function will first set NULL to X if it is not located in rank.source, then called `comm.load.balance()` to redistributed the one located in rank.source to all other ranks.

## Value

A X.gbd will be returned.

## Author(s)

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## See Also

`comm.load.balance()`, `comm.read.table()` and `comm.write.table()`.

## Examples

```
### Save code in a file "demo.r" and run with 4 processors by
### SHELL> mpiexec -np 4 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
X <- matrix(1:15, ncol = 3)
X.gbd <- comm.as.gbd(X)
comm.print(X.gbd, all.rank = TRUE)

### Finish.
finalize()
"
pbdMPI::execmpi(spmd.code, nranks = 4L)
```

---

 global balanc

*Global Balance Functions*


---

## Description

These functions are global balance methods for `gbd` data.frame (or matrix) distributed in row blocks.

## Usage

```
comm.balance.info(X.gbd, balance.method = .pbd_env$SPMD.IO$balance.method[1],
                  comm = .pbd_env$SPMD.CT$comm)
comm.load.balance(X.gbd, bal.info = NULL,
                  balance.method = .pbd_env$SPMD.IO$balance.method[1],
                  comm = .pbd_env$SPMD.CT$comm)
comm.unload.balance(new.X.gbd, bal.info, comm = .pbd_env$SPMD.CT$comm)
```

## Arguments

<code>X.gbd</code>	a <code>gbd</code> data.frame (or matrix).
<code>balance.method</code>	a balance method.
<code>bal.info</code>	a balance information returned from <code>comm.balance.info()</code> . If NULL, then this will be generated inside <code>comm.load.balance()</code> .
<code>new.X.gbd</code>	a new <code>gbd</code> of <code>X.gbd</code> (may be generated from <code>comm.load.balance()</code> ).
<code>comm</code>	a communicator number.

## Details

A typical use is to balance an input dataset `X.gbd` from `comm.read.table()`. Since by default, a two dimension data.frame is distributed in row blocks, but each processor (rank) may not (or closely) have the same number of rows. These functions redistribute the data.frame (and maybe matrix) according to the specified way in `bal.info`.

Currently, there are three balance methods are supported, `block` (uniform distributed but favor higher ranks), `block0` (as `block` but favor lower ranks), and `block.cyclic` (as `block` cyclic with one big block in one cycle).

## Value

`comm.balance.info()` returns a list containing balance information based on the input `X.gbd` and `balance.method`.

`comm.load.balance()` returns a new `gbd` data.frame (or matrix).

`comm.unload.balance()` also returns the new `gbd` data.frame back to the original `X.gbd`.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[comm.read.table\(\)](#), [comm.write.table\(\)](#), and [comm.as.gbd\(\)](#).

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 4 processors by
### SHELL> mpiexec -np 4 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))

### Get two gbd row-block data.frame.
da.block <- iris[get.jid(nrow(iris), method = \"block\"),]
da.block0 <- iris[get.jid(nrow(iris), method = \"block0\"),]

### Load balance one and unload it.
bal.info <- comm.balance.info(da.block0)
da.new <- comm.load.balance(da.block0)
da.org <- comm.unload.balance(da.new, bal.info)

### Check if all are equal.
comm.print(c(sum(da.new != da.block), sum(da.org != da.block0)),
           all.rank = TRUE)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 4L)

## End(Not run)
```

**Description**

These functions are global base functions applying on distributed data for all ranks.

**Usage**

```

comm.length(x, comm = .pbd_env$SPMD.CT$comm)
comm.sum(..., na.rm = TRUE, comm = .pbd_env$SPMD.CT$comm)
comm.mean(x, na.rm = TRUE, comm = .pbd_env$SPMD.CT$comm)
comm.var(x, na.rm = TRUE, comm = .pbd_env$SPMD.CT$comm)
comm.sd(x, na.rm = TRUE, comm = .pbd_env$SPMD.CT$comm)

```

**Arguments**

x	a vector.
...	as in sum().
na.rm	logical, if remove NA and NaN.
comm	a communicator number.

**Details**

These functions will apply globally length(), sum(), mean(), var(), and sd().

**Value**

The global values are returned to all ranks.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```

## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
if(comm.size() != 2){
  comm.cat("\n2 processors are required.\n", quiet = TRUE)
  finalize()
}

### Examples.
a <- 1:(comm.rank() + 1)

b <- comm.length(a)

```

```

comm.print(b)
b <- comm.sum(a)
comm.print(b)
b <- comm.mean(a)
comm.print(b)
b <- comm.var(a)
comm.print(b)
b <- comm.sd(a)
comm.print(b)

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)

```

---

global distance function

*Global Distance for Distributed Matrices*

---

## Description

These functions globally compute distance for all ranks.

## Usage

```

comm.dist(X.gbd, method = "euclidean", diag = FALSE, upper = FALSE,
          p = 2, comm = .pbd_env$SPMD.CT$comm,
          return.type = c("common", "gbd"))

```

## Arguments

<code>X.gbd</code>	a gbd matrix.
<code>method</code>	as in <code>dist()</code> .
<code>diag</code>	as in <code>dist()</code> .
<code>upper</code>	as in <code>dist()</code> .
<code>p</code>	as in <code>dist()</code> .
<code>comm</code>	a communicator number.
<code>return.type</code>	returning type for the distance.

**Details**

The distance function is implemented for a distributed matrix.

The return type `common` is only useful when the number of rows of the matrix is small since the returning matrix is  $N * N$  for every rank where  $N$  is the total number of rows of `X.gbd` of all ranks.

The return type `gbd` returns a `gbd` matrix (distributed across all ranks, and the `gbd` matrix has 3 columns, named "i", "j", and "value", where (i, j) is the global indices of the i-th and j-th rows of `X.gbd`, and value is the corresponding distance. The (i, j) is ordered as a distance matrix.

**Value**

A full distance matrix is returned from the `common` return type. Suppose `N.gbd` is total rows of `X.gbd`, then the distance will have  $N.gbd * (N.gbd - 1) / 2$  elements and the distance matrix will have  $N.gbd^2$  elements.

A `gbd` distance matrix with 3 columns is returned from the `gbd` return type.

**Warning**

The distance or distance matrix could be huge.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

`comm.allpairs()` and `comm.pairwise()`.

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
comm.set.seed(123456, diff = TRUE)

X.gbd <- matrix(runif(6), ncol = 3)
dist.X.common <- comm.dist(X.gbd)
dist.X.gbd <- comm.dist(X.gbd, return.type = \"gbd\")
```

```

### Verify.
dist.X <- dist(do.call(\rbind\", allgather(X.gbd)))
comm.print(all(dist.X == dist.X.common))

### Verify 2.
dist.X.df <- do.call(\rbind\", allgather(dist.X.gbd))
comm.print(all(dist.X == dist.X.df[, 3]))
comm.print(dist.X)
comm.print(dist.X.df)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)

```

---

global match.arg      *Global Argument Matching*

---

## Description

A binding for `match.arg()` that uses `comm.stop()` rather so that the error message (if there is one) is managed according to the rules of `.pbd_env$SPMD.CT`.

## Usage

```

comm.match.arg(arg, choices, several.ok=FALSE, ...,
               all.rank = .pbd_env$SPMD.CT$print.all.rank,
               rank.print = .pbd_env$SPMD.CT$rank.source,
               comm = .pbd_env$SPMD.CT$comm,
               mpi.finalize = .pbd_env$SPMD.CT$mpi.finalize,
               quit = .pbd_env$SPMD.CT$quit)

```

## Arguments

<code>arg, choices, several.ok</code>	see <code>match.arg()</code>
<code>...</code>	ignored.
<code>all.rank</code>	if all ranks print (default = FALSE).
<code>rank.print</code>	rank for printing if not all ranks print (default = 0).
<code>comm</code>	communicator for printing (default = 1).
<code>mpi.finalize</code>	if MPI should be shutdown.
<code>quit</code>	if quit R when errors happen.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

---

global pairwise

*Global Pairwise Evaluations*

---

**Description**

This function provides global pairwise evaluations.

**Usage**

```
comm.pairwise(X, pairid.gbd = NULL,
  FUN = function(x, y, ...){ return(as.vector(dist(rbind(x, y), ...))) },
  ..., diag = FALSE, symmetric = TRUE, comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

<code>X</code>	a common matrix across ranks, or a gbd matrix. (See details.)
<code>pairid.gbd</code>	a pair-wise id in a gbd format. (See details.)
<code>FUN</code>	a function to be evaluated for given pairs.
<code>...</code>	extra variables for FUN.
<code>diag</code>	if matching the same elements, (i, i) for all i.
<code>symmetric</code>	if matching upper triangular elements. TRUE for i >= j only, otherwise for all (i, j).
<code>comm</code>	a communicator number.

**Details**

This function evaluates the objective function  $FUN(X[i, ], X[j, ])$  (usually distance of two elements) on any given pair (i, j) of a matrix X.

The input X should be in common across all ranks if `pairid.gbd` is provided, e.g. from `comm.pairwise()`. i.e. X is exactly the same in every ranks, but `pairid.gbd` is different and in gbd format indicating the row pair (i, j) should be evaluated. The returning gbd matrix is ordered and indexed by `pairid.gbd`.

Note that checking consistence of X across all ranks is not implemented within this function since that drops performance and may be not accurate.

The input X should be a gbd format in row major blocks (i.e. X.gbd) if `pairid.gbd` is NULL. A internal pair indices will be built implicitly for evaluation. The returning gbd matrix is ordered and indexed by X.gbd.



**Value**

This function returns a common matrix with 3 columns named *i*, *j*, and *value*. Each value is the returned value and computed by `FUN(X[i,], X[j,])` where (*i*, *j*) is the global index as ordered in a distance matrix for *i*-th row and *j*-th columns.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[comm.pairwise\(\)](#), and [comm.dist\(\)](#).

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

smpd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
comm.set.seed(123456, diff = FALSE)
X <- matrix(rnorm(10), ncol = 2)
id.matrix <- comm.allpairs(nrow(X))

### Method original.
dist.org <- dist(X)

### Method 1.
dist.common <- comm.pairwise(X, pairid.gbd = id.matrix)

### Method 2.
# if(comm.rank() != 0){
#   X <- matrix(0, nrow = 0, ncol = 4)
# }
X.gbd <- comm.as.gbd(X)    ### The other way.
dist.gbd <- comm.pairwise(X.gbd)

### Verify.
d.org <- as.vector(dist.org)
d.1 <- do.call("\nc", allgather(dist.common[, 3]))
d.2 <- do.call("\nc", allgather(dist.gbd[, 3]))
```

```

comm.print(all(d.org == d.1))
comm.print(all(d.org == d.2))

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)

```

---

global print and cat    *Global Print and Cat Functions*

---

### Description

The functions globally print or cat a variable from specified processors, by default messages is shown on screen.

### Usage

```

comm.print(x, all.rank = .pbd_env$SPMD.CT$print.all.rank,
           rank.print = .pbd_env$SPMD.CT$rank.source,
           comm = .pbd_env$SPMD.CT$comm,
           quiet = .pbd_env$SPMD.CT$print.quiet,
           flush = .pbd_env$SPMD.CT$msg.flush,
           barrier = .pbd_env$SPMD.CT$msg.barrier,
           con = stdout(), ...)

comm.cat(..., all.rank = .pbd_env$SPMD.CT$print.all.rank,
         rank.print = .pbd_env$SPMD.CT$rank.source,
         comm = .pbd_env$SPMD.CT$comm,
         quiet = .pbd_env$SPMD.CT$print.quiet, sep = " ", fill = FALSE,
         labels = NULL, append = FALSE, flush = .pbd_env$SPMD.CT$msg.flush,
         barrier = .pbd_env$SPMD.CT$msg.barrier, con = stdout())

```

### Arguments

x	a variable to be printed.
...	variables to be cat.
all.rank	if all ranks print (default = FALSE).
rank.print	rank for printing if not all ranks print (default = 0).
comm	communicator for printing (default = 1).
quiet	FALSE for printing rank number.
sep	sep argument as in the cat() function.
fill	fill argument as in the cat() function.

labels	labels argument as in the cat() function.
append	labels argument as in the cat() function.
flush	if flush con.
barrier	if barrier con.
con	stdout() is the default to print message.

### Details

**Warning:** These two functions use barrier() to make sure the well printing process on screen, so should be called by all processors to avoid a deadlock. A typical misuse is called inside a condition check, such as `if(.comm.rank == 0) comm.cat(...)`.

`rank.print` can be a integer vector containing the ranks of processors which print messages.

### Value

A `print()` or `cat()` is called for the specified processors and the messages of the input variables is shown on screen by default.

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### Examples

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Example.
comm.print(comm.rank(), rank.print = 1)

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)
```

---

global range, max, and min

*Global Range, Max, and Min Functions*

---

## Description

These functions are global range, max and min applying on distributed data for all ranks.

## Usage

```
comm.range(..., na.rm = FALSE, comm = .pbd_env$SPMD.CT$comm)
comm.max(..., na.rm = FALSE, comm = .pbd_env$SPMD.CT$comm)
comm.min(..., na.rm = FALSE, comm = .pbd_env$SPMD.CT$comm)
```

## Arguments

...	an 'numeric' objects.
na.rm	if NA removed or not.
comm	a communicator number.

## Details

These functions will apply `range()`, `max()` and `min()` locally, and apply `allgather` to get all local results from other ranks, then apply `range()`, `max()` and `min()` on all local results.

## Value

The global values (range, max, or min) are returned to all ranks.

## Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## Examples

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
```

```

if(comm.size() != 2){
  comm.cat("\n2 processors are required.\n", quiet = TRUE)
  finalize()
}

### Examples.
a <- 1:(comm.rank() + 1)

b <- comm.range(a)
comm.print(b)
b <- comm.max(a)
comm.print(b)
b <- comm.min(a)
comm.print(b)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)

```

---

global reading

*Global Reading Functions*


---

## Description

These functions are global reading from specified file.

## Usage

```

comm.read.table(file, header = FALSE, sep = "", quote = "\"\"",
  dec = ".",
  na.strings = "NA", colClasses = NA, nrow = -1, skip = 0,
  check.names = TRUE, fill = !blank.lines.skip,
  strip.white = FALSE,
  blank.lines.skip = TRUE, comment.char = "#",
  allowEscapes = FALSE,
  flush = FALSE,
  fileEncoding = "", encoding = "unknown",
  read.method = .pbd_env$SPMD.IO$read.method[1],
  balance.method = .pbd_env$SPMD.IO$balance.method[1],
  comm = .pbd_env$SPMD.CT$comm)

```

```

comm.read.csv(file, header = TRUE, sep = ",", quote = "\"\"",
  dec = ".", fill = TRUE, comment.char = "", ...,
  read.method = .pbd_env$SPMD.IO$read.method[1],
  balance.method = .pbd_env$SPMD.IO$balance.method[1],
  comm = .pbd_env$SPMD.CT$comm)

```

```
comm.read.csv2(file, header = TRUE, sep = ";", quote = "\"",
               dec = ",", fill = TRUE, comment.char = "", ...,
               read.method = .pbd_env$SPMD.IO$read.method[1],
               balance.method = .pbd_env$SPMD.IO$balance.method[1],
               comm = .pbd_env$SPMD.CT$comm)
```

### Arguments

file	as in read.table().
header	as in read.table().
sep	as in read.table().
quote	as in read.table().
dec	as in read.table().
na.strings	as in read.table().
colClasses	as in read.table().
nrows	as in read.table().
skip	as in read.table().
check.names	as in read.table().
fill	as in read.table().
strip.white	as in read.table().
blank.lines.skip	as in read.table().
comment.char	as in read.table().
allowEscapes	as in read.table().
flush	as in read.table().
fileEncoding	as in read.table().
encoding	as in read.table().
...	as in read.csv*().
read.method	either "gbd" or "common".
balance.method	balance method for read.method = "gbd" as nrows = -1 and skip = 0 are set.
comm	a communicator number.

### Details

These functions will apply read.table() locally and sequentially from rank 0, 1, 2, ...

By default, rank 0 reads the file only, then scatter to other ranks for small datasets (.pbd\_env\$SPMD.IO\$max.read.size) in read.method = "gbd". (bcast to others in read.method = "common".)

As dataset size increases, the reading is performed from each ranks and read portion of rows in "gbd" format as described in **pbdDEMO** vignettes and used in **pmclust**.

`comm.load.balance()` is called for "gbd" method as as nrows = -1 and skip = 0 are set. Note that the default method "block" is the better way for performance in general that distributes equally and leaves residuals on higher ranks evenly. "block0" is the other way around. "block.cyclic" is only useful for converting to ddmatrix as in **pbdDMAT**.

**Value**

A distributed data.frame is returned.

All factors are disabled and read as characters or as what data should be.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[comm.load.balance\(\)](#) and [comm.write.table\(\)](#)

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 4 processors by
### SHELL> mpiexec -np 4 Rscript demo.r

smpd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))

### Check.
if(comm.size() != 4){
  comm.stop("\n4 processors are required.\n")
}

### Manually distributed iris.
da <- iris[get.jid(nrow(iris)),]

### Dump data.
comm.write.table(da, file = "iris.txt", quote = FALSE, sep = "\\t",
  row.names = FALSE)

### Read back in.
da.gbd <- comm.read.table("iris.txt", header = TRUE, sep = "\\t",
  quote = "\\")
comm.print(c(nrow(da), nrow(da.gbd)), all.rank = TRUE)

### Read in common.
da.common <- comm.read.table("iris.txt", header = TRUE, sep = "\\t",
  quote = "\\", read.method = "common")
comm.print(c(nrow(da.common), sum(da.common != iris)))

### Finish.
finalize()
```

```

"
# execmpi(spmd.code, nranks = 4L)

## End(Not run)

```

---

global Rprof

*A Rprof Function for SPMD Routines*


---

## Description

A Rprof function for use with parallel codes executed in the batch SPMD style.

## Usage

```

comm.Rprof(filename = "Rprof.out", append = FALSE, interval = 0.02,
            memory.profiling = FALSE, gc.profiling = FALSE,
            line.profiling = FALSE, numfiles = 100L, bufsize = 10000L,
            all.rank = .pbd_env$SPMD.CT$Rprof.all.rank,
            rank.Rprof = .pbd_env$SPMD.CT$rank.source,
            comm = .pbd_env$SPMD.CT$comm)

```

## Arguments

filename	as in <a href="#">Rprof()</a> .
append	as in <a href="#">Rprof()</a> .
interval	as in <a href="#">Rprof()</a> .
memory.profiling	as in <a href="#">Rprof()</a> .
gc.profiling	as in <a href="#">Rprof()</a> .
line.profiling	as in <a href="#">Rprof()</a> .
numfiles	as in <a href="#">Rprof()</a> .
bufsize	as in <a href="#">Rprof()</a> .
all.rank	if calling Rprof on all ranks (default = FALSE).
rank.Rprof	rank for calling Rprof if all.rank = FALSE (default = 0).
comm	a communicator number.

## Details

as in [Rprof\(\)](#).

## Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.



**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

---

global sort

*Global Quick Sort for Distributed Vectors or Matrices*

---

**Description**

This function globally sorts distributed data for all ranks.

**Usage**

```
comm.sort(x, decreasing = FALSE, na.last = NA,  
          comm = .pbd_env$SPMD.CT$comm,  
          status = .pbd_env$SPMD.CT$status)
```

**Arguments**

x	a vector.
decreasing	logical. Should the sort order be increasing or decreasing?
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
comm	a communicator number.
status	a status number.

**Details**

The distributed quick sort is implemented for this functions.

**Value**

The returns are the same size of x but in global sorting order.

**Warning**

All ranks may not have a NULL x.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```

## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
comm.set.seed(123456, diff = TRUE)
x <- c(rnorm(5 + .comm.rank * 2), NA)
# x <- sample(1:5, 5 + .comm.rank * 2, replace = TRUE)
comm.end.seed()

if(.comm.rank == 1){
  x <- NULL    ### Test for NULL or 0 vector
}

y <- allgather(x)
comm.print(y)

y <- comm.sort(x)
y <- allgather(y)
comm.print(y)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)

```

---

global stop and warning

*Global Stop and Warning Functions*

---

**Description**

These functions are global stop and warning applying on distributed data for all ranks, and are called by experts only. These functions may lead to potential performance degradation and system termination.

**Usage**

```
comm.stop(..., call. = TRUE, domain = NULL,
```

```

all.rank = .pbd_env$SPMD.CT$print.all.rank,
rank.print = .pbd_env$SPMD.CT$rank.source,
comm = .pbd_env$SPMD.CT$comm,
mpi.finalize = .pbd_env$SPMD.CT$mpi.finalize,
quit = .pbd_env$SPMD.CT$quit)

comm.warning(..., call. = TRUE, immediate. = FALSE, domain = NULL,
  all.rank = .pbd_env$SPMD.CT$print.all.rank,
  rank.print = .pbd_env$SPMD.CT$rank.source,
  comm = .pbd_env$SPMD.CT$comm)

comm.warnings(...,
  all.rank = .pbd_env$SPMD.CT$print.all.rank,
  rank.print = .pbd_env$SPMD.CT$rank.source,
  comm = .pbd_env$SPMD.CT$comm)

comm.stopifnot(..., call. = TRUE, domain = NULL,
  all.rank = .pbd_env$SPMD.CT$print.all.rank,
  rank.print = .pbd_env$SPMD.CT$rank.source,
  comm = .pbd_env$SPMD.CT$comm,
  mpi.finalize = .pbd_env$SPMD.CT$mpi.finalize,
  quit = .pbd_env$SPMD.CT$quit)

```

## Arguments

<code>...</code>	variables to be cat.
<code>call.</code>	see <code>stop()</code> and <code>warnings()</code> .
<code>immediate.</code>	see <code>stop()</code> and <code>warnings()</code> .
<code>domain</code>	see <code>stop()</code> and <code>warnings()</code> .
<code>all.rank</code>	if all ranks print (default = <code>FALSE</code> ).
<code>rank.print</code>	rank for printing if not all ranks print (default = 0).
<code>comm</code>	communicator for printing (default = 1).
<code>mpi.finalize</code>	if MPI should be shutdown.
<code>quit</code>	if quit R when errors happen.

## Details

These functions will respectively apply `stop()`, `warning()`, `warnings()`, and `stopifnot()` locally.

## Value

`comm.stop()` and `comm.stopifnot()` terminate all ranks, `comm.warning()` returns messages, and `comm.warnings()` print the message.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
if(comm.size() != 2){
  comm.cat("\n2 processors are required.\n", quiet = TRUE)
  finalize()
}

### Examples.
comm.warning("\ntest warning.\n")
comm.warnings()
comm.stop("\ntest stop.\n")
comm.stopifnot(1 == 2)

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)
```

---

global timer

*A Timing Function for SPMD Routines*

---

**Description**

A timing function for use with parallel codes executed in the batch SPMD style.

**Usage**

```
comm.timer(timed, comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

timed	expression to be timed.
comm	a communicator number.

**Details**

Finds the min, mean, and max execution time across all independent processes executing the operation timed.

**Author(s)**

Drew Schmidt.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

---

global which, which.max, and which.min  
*Global Which Functions*

---

**Description**

These functions are global which, which.max and which.min applying on distributed data for all ranks.

**Usage**

```
comm.which(x, arr.ind = FALSE, useNames = TRUE,  
           comm = .pbd_env$SPMD.CT$comm)  
comm.which.max(x, comm = .pbd_env$SPMD.CT$comm)  
comm.which.min(x, comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

x	a 'logical' vector or array as in which(), or an 'numeric' objects in which.max() and which.min().
arr.ind	logical, as in which().
useNames	logical, as in which().
comm	a communicator number.

**Details**

These functions will apply which(), which.max() and which.min() locally, and apply allgather() to get all local results from other ranks.

**Value**

The global values (`which()`, `which.max()`, or `which.min()`) are returned to all ranks.

`comm.which()` returns with two columns, 'rank id' and 'index of TRUE'.

`comm.which.max()` and `comm.which.min()` return with three values, 'the `_smallest_` rank id', 'index of the `_first_` maximum or minimum', and 'max/min value of x'.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[comm.read.table\(\)](#)

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
if(comm.size() != 2){
  comm.cat("\n2 processors are required.\n", quiet = TRUE)
  finalize()
}

### Examples.
a <- 1:(comm.rank() + 1)

b <- comm.which(a == 2)
comm.print(b)
b <- comm.which.max(a)
comm.print(b)
b <- comm.which.min(a)
comm.print(b)

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)
```

**Description**

These functions are global writing applying on distributed data for all ranks.

**Usage**

```
comm.write(x, file = "data", ncolumns = if(is.character(x)) 1 else 5,
           append = FALSE, sep = " ", comm = .pbd_env$SPMD.CT$comm)
comm.write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
                eol = "\n", na = "NA", dec = ".", row.names = TRUE,
                col.names = TRUE, qmethod = c("escape", "double"),
                fileEncoding = "", comm = .pbd_env$SPMD.CT$comm)

comm.write.csv(..., comm = .pbd_env$SPMD.CT$comm)
comm.write.csv2(..., comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

x	as in write() or write.table().
file	as in write() or write.table().
ncolumns	as in write*().
append	as in write*().
sep	as in write*().
quote	as in write*().
eol	as in write*().
na	as in write*().
dec	as in write*().
row.names	as in write*().
col.names	as in write*().
qmethod	as in write*().
fileEncoding	as in write*().
...	as in write*().
comm	a communicator number.

**Details**

These functions will apply write\*() locally and sequentially from rank 0, 1, 2, ...

By default, rank 0 makes the file, and rest of ranks append the data.

**Value**

A file will be returned.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[comm.load.balance\(\)](#) and [comm.read.table\(\)](#)

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
if(comm.size() != 2){
  comm.cat("\n2 processors are required.\n", quiet = TRUE)
  finalize()
}

### Examples.
comm.write((1:5) + comm.rank(), file = "test.txt")

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)
```

**Description**

The functions call MPI info functions.



**Usage**

```
info.create(info = .pbd_env$SPMD.CT$info)
info.set(info = .pbd_env$SPMD.CT$info, key, value)
info.free(info = .pbd_env$SPMD.CT$info)
info.c2f(info = .pbd_env$SPMD.CT$info)
```

**Arguments**

info	a info number.
key	a character string to be set.
value	a character string to be set associate with key.

**Details**

These functions are for internal functions. Potentially, they set info for initialization of master and workers.

**Value**

An invisible state of MPI call is returned.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
info.create(0L)
info.set(0L, \"file\", \"appschema\")
info.free(0L)

### Finish.
finalize()
```

```

"
# execmpi(spm�.code, nranks = 2L)

## End(Not run)

```

---

irecv-method

*A Rank Receives (Nonblocking) an Object from the Other Rank*


---

### Description

This method lets a rank receive (nonblocking) an object from the other rank in the same communicator. The default return is the object sent from the other rank.

### Usage

```

irecv(x.buffer = NULL, rank.source = .pbd_env$SPMD.CT$rank.source,
      tag = .pbd_env$SPMD.CT$tag, comm = .pbd_env$SPMD.CT$comm,
      request = .pbd_env$SPMD.CT$request,
      status = .pbd_env$SPMD.CT$status)

```

### Arguments

<code>x.buffer</code>	a buffer to store <code>x</code> sent from the other rank.
<code>rank.source</code>	a source rank where <code>x</code> sent from
<code>tag</code>	a tag number.
<code>comm</code>	a communicator number.
<code>request</code>	a request number.
<code>status</code>	a status number.

### Details

A corresponding `send()/isend()` should be evoked at the corresponding rank `rank.source`.

**Warning:** `irecv()` is not safe for R since R is not a thread safe package that a dynamic returning object requires certain blocking or barrier at some where. Current, the default method is equivalent to the default method of `recv()`.

### Value

An object is returned by default.

### Methods

For calling `spm�.irecv.*()`:

```

signature(x = "ANY")
signature(x = "integer")
signature(x = "numeric")
signature(x = "raw")

```

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[recv\(\)](#), [send\(\)](#), [isend\(\)](#).

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
if(.comm.rank == 0){
  y <- send(matrix(x, nrow = 1))
} else if(.comm.rank == 1){
  y <- irecv()
}
comm.print(y, rank.print = 1)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)
```

---

is.comm.null

*Check if a MPI\_COMM\_NULL*

---

**Description**

The functions check MPI\_COMM\_NULL.

**Usage**

```
is.comm.null(comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

comm                    a comm number.

**Details**

These functions are for internal uses.

**Value**

TRUE if input comm is MPI\_COMM\_NULL, otherwise FALSE.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
is.comm.null(0L)
is.comm.null(1L)

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)
```

---

isend-method	<i>A Rank Send (Nonblocking) an Object to the Other Rank</i>
--------------	--

---

### Description

This method lets a rank send (nonblocking) a object to the other rank in the same communicator. The default return is NULL.

### Usage

```
isend(x, rank.dest = .pbd_env$SPMD.CT$rank.dest,  
      tag = .pbd_env$SPMD.CT$tag,  
      comm = .pbd_env$SPMD.CT$comm, request = .pbd_env$SPMD.CT$request)
```

### Arguments

x	an object to be sent from a rank.
rank.dest	a rank of destination where x send to.
tag	a tag number.
comm	a communicator number.
request	a request number.

### Details

A corresponding `recv()` or `irecv()` should be evoked at the corresponding rank `rank.dest`.

### Value

A NULL is returned by default.

### Methods

For calling `spmd.isend.*()`:

```
signature(x = "ANY")  
signature(x = "integer")  
signature(x = "numeric")  
signature(x = "raw")
```

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

`send()`, `recv()`, `irecv()`.

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
if(.comm.rank == 0){
  y <- isend(matrix(x, nrow = 1))
} else if(.comm.rank == 1){
  y <- recv()
}
comm.print(y, rank.print = 1)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)
```

---

MPI array pointers

*Set or Get MPI Array Pointers in R*

---

**Description**

The function `set/get` a point address in R where the point point to a structure containing MPI arrays.

**Usage**

```
arrange.mpi.apts()
```

**Details**

Since `Rmpi/pbdMPI` use pre-allocate memory to store `comm`, `status`, `datatype`, `info`, `request`, this function provides a variable in R to let different APIs share the same memory address.

If the package loads first, then this sets ‘`._MPI_APTS_`’ in the `.GlobalEnv` of R. If the package does not load before other MPI APIs, then this points an structure point to the external memory according to ‘`._MPI_APTS_`’, i.e. pre-set by other MPI APIs.

`pbdMPI/R/arrange.mpi.apts` provides the R code, and `pbdMPI/src/pkg_*.c` provides the details of this call.

### Value

‘`._MPI_APTS_`’ is set in the `.GlobalEnv` of R.

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### Examples

```
## Not run:
### See source code for the details.

## End(Not run)
```

---

probe

*Probe Functions*

---

### Description

The functions call MPI probe functions.

### Usage

```
probe(rank.source = .pbd_env$SPMD.CT$rank.source,
      tag = .pbd_env$SPMD.CT$tag, comm = .pbd_env$SPMD.CT$comm,
      status = .pbd_env$SPMD.CT$status)
iprobe(rank.source = .pbd_env$SPMD.CT$rank.source,
       tag = .pbd_env$SPMD.CT$tag, comm = .pbd_env$SPMD.CT$comm,
       status = .pbd_env$SPMD.CT$status)
```

### Arguments

<code>rank.source</code>	a source rank where an object sent from.
<code>tag</code>	a tag number.
<code>comm</code>	a communicator number.
<code>status</code>	a status number.

**Details**

These functions are for internal functions. Potentially, they set/get probe for receiving data.

**Value**

An invisible state of MPI call is returned.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```
## Not run:  
### See source code of spmd.recv.default() for an example.  
  
## End(Not run)
```

---

recv-method

*A Rank Receives (Blocking) an Object from the Other Rank*

---

**Description**

This method lets a rank receive (blocking) an object from the other rank in the same communicator. The default return is the object sent from the other rank.

**Usage**

```
recv(x.buffer = NULL, rank.source = .pbd_env$SPMD.CT$rank.source,  
     tag = .pbd_env$SPMD.CT$tag, comm = .pbd_env$SPMD.CT$comm,  
     status = .pbd_env$SPMD.CT$status)
```

**Arguments**

x.buffer	a buffer to store x sent from the other rank.
rank.source	a source rank where x sent from
tag	a tag number.
comm	a communicator number.
status	a status number.



**Details**

A corresponding `send()` should be evoked at the corresponding rank `rank.source`.

**Value**

An object is returned by default.

**Methods**

For calling `spmd.recv.*()`:

```
signature(x = "ANY")
```

```
signature(x = "integer")
```

```
signature(x = "numeric")
```

```
signature(x = "raw")
```

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[irecv\(\)](#), [send\(\)](#), [isend\(\)](#).

**Examples**

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r
```

```
spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()
```

```
### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
if(.comm.rank == 0){
  y <- send(matrix(x, nrow = 1))
} else if(.comm.rank == 1){
  y <- recv()
}
```

```

comm.print(y, rank.print = 1)

### Finish.
finalize()
"
pbdMPI::execmpi(spmc.code, nrank = 2L)

```

---

reduce-method

*A Rank Receive a Reduction of Objects from Every Rank*


---

### Description

This method lets a rank receive a reduction of objects from every rank in the same communicator based on a given operation. The default return is an object as the input.

### Usage

```

reduce(x, x.buffer = NULL, op = .pbd_env$SPMD.CT$op,
       rank.dest = .pbd_env$SPMD.CT$rank.source,
       comm = .pbd_env$SPMD.CT$comm)

```

### Arguments

<code>x</code>	an object to be gathered from all ranks.
<code>x.buffer</code>	a buffer to hold the return object which probably has <code>x</code> with the same type of <code>x</code> .
<code>op</code>	a reduction operation applied on combine all <code>x</code> .
<code>rank.dest</code>	a rank of destination where all <code>x</code> reduce to.
<code>comm</code>	a communicator number.

### Details

By default, the object is reduced to `.pbd_env$SPMD.CT$rank.source`, i.e. *rank 0L*.

All `x` on all ranks are likely presumed to have the same size and type.

`x.buffer` can be `NULL` or unspecified. If specified, the type should be either integer or double specified correctly according to the type of `x`.

### Value

The reduced object of the same type as `x` is returned by default.

## Methods

For calling `spmd.reduce.*()`:

```
signature(x = "ANY", x.buffer = "missing")
signature(x = "integer", x.buffer = "integer")
signature(x = "numeric", x.buffer = "numeric")
signature(x = "logical", x.buffer = "logical")
signature(x = "float32", x.buffer = "float32")
```

## Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## See Also

[allgather\(\)](#), [gather\(\)](#), [reduce\(\)](#).

## Examples

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
y <- reduce(matrix(x, nrow = 1), op = \"sum\")
comm.print(y)

y <- reduce(x, double(N), op = \"prod\")
comm.print(y)

x <- as.logical(round(runif(N)))
y <- reduce(x, logical(N), op = \"land\")
comm.print(y)

### Finish.
finalize()
```

```
"
pbdMPI::execmpi(spm�.code = spm�.code, nranks = 2L)
```

---

scatter-method

*A Rank Scatter Objects to Every Rank*


---

### Description

This method lets a rank scatter objects to every rank in the same communicator. The default input is a list of length equal to ‘comm size’ and the default return is an element of the list.

### Usage

```
scatter(x, x.buffer = NULL, x.count = NULL, displs = NULL,
       rank.source = .pbd_env$SPMD.CT$rank.source,
       comm = .pbd_env$SPMD.CT$comm)
```

### Arguments

<code>x</code>	an object of length ‘comm size’ to be scattered to all ranks.
<code>x.buffer</code>	a buffer to hold the return object which probably has ‘size of element of x’ with the same type of the element of x.
<code>x.count</code>	a vector of length ‘comm size’ containing all object lengths.
<code>displs</code>	<code>c(0L, cumsum(x.count))</code> by default.
<code>rank.source</code>	a rank of source where elements of x scatter from.
<code>comm</code>	a communicator number.

### Details

All elements of `x` are likely presumed to have the same size and type.

`x.buffer`, `x.count`, and `displs` can be `NULL` or unspecified. If specified, the type should be one of integer, double, or raw specified correctly according to the type of `x`.

If `x.count` is specified, then the `spm�.scatterv.*()` is called.

### Value

An element of `x` is returned according to the rank id.

**Methods**

For calling `spmd.scatter.*()`:

```
signature(x = "ANY", x.buffer = "missing", x.count = "missing")
signature(x = "integer", x.buffer = "integer", x.count = "missing")
signature(x = "numeric", x.buffer = "numeric", x.count = "missing")
signature(x = "raw", x.buffer = "raw", x.count = "missing")
```

For calling `spmd.scatterv.*()`:

```
signature(x = "ANY", x.buffer = "missing", x.count = "integer")
signature(x = "ANY", x.buffer = "ANY", x.count = "integer")
signature(x = "integer", x.buffer = "integer", x.count = "integer")
signature(x = "numeric", x.buffer = "numeric", x.count = "integer")
signature(x = "raw", x.buffer = "raw", x.count = "integer")
```

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[bcast\(\)](#).

**Examples**

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- split(1:(N * .comm.size), rep(1:.comm.size, N))
y <- scatter(lapply(x, matrix, nrow = 1))
comm.print(y)
y <- scatter(x, double(N))
comm.print(y)
```

```

### Finish.
finalize()
"
pbdMPI::execmpi(spmd.code, nrank = 2L)

```

---

seed for RNG

*Seed Functions for Random Number Generators*


---

## Description

These functions set/end/reset seeds to all ranks. By default, these functions are wrappers of **rlecuyer** which implements the random number generator with multiple independent streams developed by L'Ecuyer et al (2002).

## Usage

```

comm.set.seed(seed, diff = FALSE, state = NULL,
              comm = .pbd_env$SPMD.CT$comm)
comm.seed.state(comm = .pbd_env$SPMD.CT$comm)
comm.end.seed(comm = .pbd_env$SPMD.CT$comm)
comm.reset.seed(comm = .pbd_env$SPMD.CT$comm)

```

## Arguments

seed	one integer or six integers as in <b>rlecuyer</b> .
diff	if all ranks use the same stream. (default = FALSE)
state	a new state to overwrite seed.
comm	a communicator number.

## Details

`comm.set.seed()` sets the given seed to all ranks. If `diff = FALSE`, then all ranks generate one stream and use that stream. Otherwise, all ranks generate `COMM.SIZE` streams and use the stream named by `COMM.RANK`.

Also, `comm.set.seed()` can assign to arbitrarily state obtained from `comm.seed.state()`.

`comm.seed.state()` obtains current state of seed which ends the stream first (update state), gets the state, and continues the stream (pretend as nothing happens).

`comm.end.seed()` ends and deletes seed from all ranks.

`comm.reset.seed()` resets seed to initial start steps which end the current seed and reset everything back to the start stream. Use this function with caution.

## Value

Several hidden objects are set in the `.GlobalEnv`, see **rlecuyer** package for details.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Pierre L'Ecuyer, Simard, R., Chen, E.J., and Kelton, W.D. (2002) An Object-Oriented Random-Number Package with Many Long Streams and Substreams. *Operations Research*, 50(6), 1073-1075.

<http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf>

Sevcikova, H. and Rossini, T. (2012) rlecuyer: R interface to RNG with multiple streams. R Package, URL <https://cran.r-project.org/package=rlecuyer>

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

`.lec.SetPackageSeed()`, `.lec.CreateStream()`, `.lec.CurrentStream()`, `.lec.CurrentStreamEnd()`, `.lec.DeleteStream()`, `.lec.SetSeed()`, and `.lec.GetState()`.

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

smd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
comm.set.seed(123456)
comm.print(runif(5), all.rank = TRUE)
comm.reset.seed()
comm.print(runif(5), all.rank = TRUE)
comm.end.seed()

### Obtain the seed state.
comm.set.seed(123456, diff = TRUE)
comm.print(runif(5), all.rank = TRUE)
saved.seed <- comm.seed.state()  ### save the state.
comm.print(runif(5), all.rank = TRUE)
comm.end.seed()

### Start from a saved state.
comm.set.seed(123456, state = saved.seed)  ### rewind to the state.
comm.print(runif(5), all.rank = TRUE)
comm.end.seed()

### Finish.
```

```
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)
```

---

send-method

*A Rank Send (blocking) an Object to the Other Rank*


---

### Description

This method lets a rank send (blocking) an object to the other rank in the same communicator. The default return is NULL.

### Usage

```
send(x, rank.dest = .pbd_env$SPMD.CT$rank.dest,
     tag = .pbd_env$SPMD.CT$tag,
     comm = .pbd_env$SPMD.CT$comm)
```

### Arguments

x	an object to be sent from a rank.
rank.dest	a rank of destination where x send to.
tag	a tag number.
comm	a communicator number.

### Details

A corresponding `recv()` should be evoked at the corresponding rank `rank.dest`.

### Value

A NULL is returned by default.

### Methods

For calling `spmd.send.*()`:

```
signature(x = "ANY")
signature(x = "integer")
signature(x = "numeric")
signature(x = "raw")
```

### Author(s)

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.



## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## See Also

[isend\(\)](#), [recv\(\)](#), [irecv\(\)](#).

## Examples

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
if(.comm.rank == 0){
  y <- send(matrix(x, nrow = 1))
} else if(.comm.rank == 1){
  y <- recv()
}
comm.print(y, rank.print = 1)

### Finish.
finalize()
"

pbdMPI::execmpi(spmd.code, nranks = 2L)
```

---

sendrecv-method

*Send and Receive an Object to and from Other Ranks*

---

## Description

This method lets a rank send an object to the other rank and receive an object from another rank in the same communicator. The default return is x.

## Usage

```
sendrecv(x, x.buffer = NULL,
         rank.dest = (comm.rank(.pbd_env$SPMD.CT$comm) + 1) %%
                   comm.size(.pbd_env$SPMD.CT$comm),
```

```

send.tag = .pbd_env$SPMD.CT$tag,
rank.source = (comm.rank(.pbd_env$SPMD.CT$comm) - 1) %%
              comm.size(.pbd_env$SPMD.CT$comm),
recv.tag = .pbd_env$SPMD.CT$tag,
comm = .pbd_env$SPMD.CT$comm, status = .pbd_env$SPMD.CT$status)

```

### Arguments

<code>x</code>	an object to be sent from a rank.
<code>x.buffer</code>	a buffer to store <code>x</code> sent from the other rank.
<code>rank.dest</code>	a rank of destination where <code>x</code> send to.
<code>send.tag</code>	a send tag number.
<code>rank.source</code>	a source rank where <code>x</code> sent from.
<code>recv.tag</code>	a receive tag number.
<code>comm</code>	a communicator number.
<code>status</code>	a status number.

### Details

A corresponding `sendrecv()` should be evoked at the corresponding ranks `rank.dest` and `rank.source`. `rank.dest` and `rank.source` can be `as.integer(NULL)` to create a silent `sendrecv` operation which is more efficient than setting `rank.dest` and `rank.source` to be equal.

### Value

A `x` is returned by default.

### Methods

For calling `spmd.sendrecv.*()`:

```

signature(x = "ANY", x.buffer = "ANY")
signature(x = "integer", x.buffer = "integer")
signature(x = "numeric", x.buffer = "numeric")
signature(x = "raw", x.buffer = "raw")

```

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### See Also

[sendrecv.replace\(\)](#).

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.size
y <- sendrecv(matrix(x, nrow = 1))
comm.print(y, rank.print = 1)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)
```

---

sendrecv.replace-method

*Send and Receive an Object to and from Other Ranks*


---

**Description**

This method lets a rank send an object to the other rank and receive an object from another rank in the same communicator. The default return is `x`.

**Usage**

```
sendrecv.replace(x,
  rank.dest = (comm.rank(.pbd_env$SPMD.CT$comm) + 1) %%
    comm.size(.pbd_env$SPMD.CT$comm),
  send.tag = .pbd_env$SPMD.CT$tag,
  rank.source = (comm.rank(.pbd_env$SPMD.CT$comm) - 1) %%
    comm.size(.pbd_env$SPMD.CT$comm),
  recv.tag = .pbd_env$SPMD.CT$tag,
  comm = .pbd_env$SPMD.CT$comm, status = .pbd_env$SPMD.CT$status)
```

**Arguments**

`x` an object to be sent from a rank.

<code>rank.dest</code>	a rank of destination where x send to.
<code>send.tag</code>	a send tag number.
<code>rank.source</code>	a source rank where x sent from.
<code>recv.tag</code>	a receive tag number.
<code>comm</code>	a communicator number.
<code>status</code>	a status number.

### Details

A corresponding `sendrecv.replace()` should be evoked at the corresponding ranks `rank.dest` and `rank.source`.

`rank.dest` and `rank.source` can be `as.integer(NULL)` to create a silent `sendrecv` operation which is more efficient than setting `rank.dest` and `rank.source` to be equal.

**Warning:** `sendrecv.replace()` is not safe for `R` since `R` is not a thread safe package that a dynamic returning object requires certain blocking or barrier at some where. The replaced object or memory address 'MUST' return correctly. This is almost equivalent to `sendrecv()`.

### Value

A `x` is returned by default.

### Methods

For calling `spmd.sendrecv.replace.*()`:

```
signature(x = "ANY")
```

```
signature(x = "integer")
```

```
signature(x = "numeric")
```

```
signature(x = "raw")
```

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

### See Also

[sendrecv\(\)](#).

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.size
x <- sendrecv.replace(matrix(x, nrow = 1))
comm.print(x, rank.print = 1)

### Finish.
finalize()
"
# execmpi(spmd.code, nrank = 2L)

## End(Not run)
```

---

Set global pbd options

*Set Global pbdR Options*

---

**Description**

This is an advanced function to set pbdR options.

**Usage**

```
pbd_opt(..., bytext = "", envir = .GlobalEnv)
```

**Arguments**

...	in argument format option = value to set .pbd_env\$option <- value inside the envir.
bytext	in text format "option = value" to set .pbd_env\$option <- value inside the envir.
envir	by default the global environment is used.

## Details

... allows multiple options in `envir$.pbd_env`, but only in a simple way.

`bytext` allows to assign options by text in `envir$.pbd_env`, but can assign advanced objects. For example, `"option$suboption <- value"` will set `envir$.pbd_env$option$suboption <- value`.

## Value

No value is returned.

## Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)> and Drew Schmidt.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## See Also

[.pbd\\_env](#), [SPMD.CT\(\)](#), [SPMD.OP\(\)](#), [SPMD.IO\(\)](#), [SPMD.TP\(\)](#), and [.mpiopt\\_init\(\)](#).

## Examples

```
## Not run:
### Save code in a file "demo.r" and run with 4 processors by
### SHELL> mpiexec -np 4 Rscript demo.r

### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()

### Examples.
ls(.pbd_env)
pbd_opt(ICTXT = c(2, 2))
pbd_opt(bytext = "grid.new <- list(); grid.new$ICTXT <- c(4, 4)")
pbd_opt(BLDIM = c(16, 16), bytext = "grid.new$BLDIM = c(8, 8)")
ls(.pbd_env)
.pbd_env$ICTXT
.pbd_env$BLDIM
.pbd_env$grid.new

### Finish.
finalize()

## End(Not run)
```

---

`sourcetag`*Functions to Obtain source and tag*

---

## Description

The functions extract `MPI_ANY_SOURCE`, `MPI_ANY_TAG`, `MPI_status.source` and `MPI_status.tag`.

## Usage

```
anysource()
anytag()
get.sourcetag(status = .pbd_env$SPMD.CT$status)
```

## Arguments

`status` a status number.

## Details

These functions are for internal uses.

## Value

Corresponding status will be returned.

## Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## Examples

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()
if(.comm.size < 2)
  comm.stop("\nAt least two processors are required.\n")
```

```

### Examples.
if(.comm.rank != 0){
  send(as.integer(.comm.rank * 10), rank.dest = 0L,
        tag = as.integer(.comm.rank + 10))
}
if(.comm.rank == 0){
  for(i in 1:(.comm.size - 1)){
    ret <- recv(x.buffer = integer(1),
               rank.source = anysouce(), tag = anytag())
    sourcetag <- get.sourcetag()
    print(c(sourcetag, ret))
  }
}

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)

## End(Not run)

```

---

 SPMD Control

*Sets of controls in pbdMPI.*


---

## Description

These sets of controls are used to provide default values in this package.

## Format

Objects contain several parameters for communicators and methods.

## Details

The elements of `.pbd_env$SPMD.CT` are default values for controls including

Elements	Default	Usage
<code>comm</code>	0L	a communicator index
<code>intercomm</code>	2L	an inter communicator index
<code>info</code>	0L	an info index
<code>newcomm</code>	1L	a new communicator index
<code>op</code>	"sum"	an operation
<code>port.name</code>	"spmdport"	an operation
<code>print.all.rank</code>	FALSE	if all ranks print message
<code>print.quiet</code>	FALSE	if print/cat rank information
<code>rank.root</code>	0L	a rank of root
<code>rank.source</code>	0L	a rank of source
<code>rank.dest</code>	1L	a rank of destination
<code>request</code>	0L	a request index



<code>serv.name</code>	"spmdserv"	a service name
<code>status</code>	0L	a status index
<code>tag</code>	0L	a tag number
<code>unlist</code>	FALSE	if unlist returning
<code>divide.method</code>	"block"	a way to divide jobs or data
<code>mpi.finalize</code>	TRUE	if shutdown MPI
<code>quit</code>	TRUE	if quit when errors occur
<code>msg.flush</code>	TRUE	if flush message for <code>comm.cat/comm.print</code>
<code>msg.barrier</code>	TRUE	if barrier message for <code>comm.cat/comm.print</code>
<code>Rprof.all.rank</code>	FALSE	if call <code>Rprof</code> on all ranks
<code>lazy.check</code>	TRUEE	if use lazy check on all ranks

The elements of `.pbd_env$SPMD.OP` list the implemented operations for `reduce()` and `allreduce()`. Currently, four operations are implemented "sum", "prod", "max", and "min".

The elements of `.SPMD.IO` are default values for input and output including

Elements	Default	Usage
<code>max.read.size</code>	5.2e6	max of reading size (5 MB)
<code>max.test.lines</code>	500	max of testing lines
<code>read.method</code>	"gbd"	default reading method
<code>balance.method</code>	"block"	default load balance method

where `balance.method` is only used for "gbd" reading method when `nrows = -1` and `skip = 0` are set.

The elements of `.pbd_env$SPMD.TP` are default values mainly for task pull including

Elements	Default	Usage
<code>bcast</code>	FALSE	if <code>bcase()</code> objects to all ranks
<code>barrier</code>	TRUE	if call <code>barrier()</code> for all ranks
<code>try</code>	TRUE	if use <code>try()</code> in works
<code>try.silent</code>	FALSE	if silent the <code>try()</code> message

See [task.pull\(\)](#) for details.

### Author(s)

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

### References

Programming with Big Data in R Website: <http://r-pbd.org/>

---

 SPMD Control Functions

*Sets of controls in pbdMPI.*

---

**Description**

These sets of controls are used to provide default values in this package.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[.pbd\\_env.](#)

---

 Task Pull

*Functions for Task Pull Parallelism*

---

**Description**

These functions are designed in SPMD but assuming rank 0 is a master and rests are workers.

**Usage**

```
task.pull(jids, FUN, ..., rank.master = .pbd_env$SPMD.CT$rank.root,
          comm = .pbd_env$SPMD.CT$comm, bcast = .pbd_env$SPMD.TP$bcast,
          barrier = .pbd_env$SPMD.TP$barrier,
          try = .pbd_env$SPMD.TP$try,
          try.silent = .pbd_env$SPMD.TP$try.silent)
```

```
task.pull.workers(FUN = function(jid, ...){ return(jid) }, ...,
                  rank.master = .pbd_env$SPMD.CT$rank.root,
                  comm = .pbd_env$SPMD.CT$comm,
                  try = .pbd_env$SPMD.TP$try,
                  try.silent = .pbd_env$SPMD.TP$try.silent)
```

```
task.pull.master(jids, rank.master = .pbd_env$SPMD.CT$rank.root,
                 comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

<code>jids</code>	all job ids (a vector of positive integers).
<code>FUN</code>	a function to be evaluated in workers.
<code>...</code>	extra variables for <code>FUN</code> .
<code>rank.master</code>	a rank of master where <code>jid</code> sent from.
<code>comm</code>	a communicator number.
<code>bcast</code>	if <code>bcast</code> to all ranks.
<code>barrier</code>	if barrier for all ranks.
<code>try</code>	if use <code>try()</code> to avoid breaks. CAUTION: <code>try = FALSE</code> is not safe and can stop all MPI/R jobs.
<code>try.silent</code>	if turn off the error message from <code>try()</code> .

**Details**

All of these functions are for SPMD, NOT for master/workers.

`FUN` is a user defined function which has `jid` as the first argument and other variables are given in `...`

The `jids` will be asked by workers when jobs are available and workers are no job in hand.

**Value**

A list with length `comm.size() - 1` will be returned for mater, but `NULL` for workers. Each element of the list contains returns `ret` of the `FUN` call.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**See Also**

[get.jid\(\)](#).

**Examples**

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
# mpiexec -np 4 Rscript -e "demo(task_pull,'pbdMPI',ask=F,echo=F)"
### Or
# execmpi("demo(task_pull,'pbdMPI',ask=F,echo=F)", nrank = 4L)

## End(Not run)
```

---

Utility <i>execmpi</i>	<i>Execute MPI code in system</i>
------------------------	-----------------------------------

---

**Description**

This function basically saves code in a `spmd.file` and executes MPI via R's system call e.g. `system("mpiexec -np 2 Rscript`

**Usage**

```
execmpi(spmd.code = NULL, spmd.file = NULL,
        mpicmd = NULL, nranks = 2L, verbose = TRUE)
```

**Arguments**

<code>spmd.code</code>	SPMD code to be run via <code>mpicmd</code> and <code>Rscript</code> .
<code>spmd.file</code>	a file contains SPMD code to be run via <code>mpicmd</code> and <code>Rscript</code> .
<code>mpicmd</code>	MPI executable command. If <code>NULL</code> , system default will be searched.
<code>nranks</code>	number of processes to run the SPMD code invoked by <code>mpicmd</code> .
<code>verbose</code>	print SPMD code outputs and MPI messages.

**Details**

When the `spmd.code` is `NULL`: The code should be already saved in the file named `spmd.file` for using.

When the `spmd.code` is not `NULL`: The `spmd.code` will be dumped to a temp file (`spmd.file`) via the call `writeLines(spmd.code, conn)` where `conn <- file(spmd.file, open = "wt")`. The file will be closed after the dumping.

When `spmd.file` is ready (either dumped from `spmd.code` or provided by the user), the steps below will be followed: If `spmd.file = NULL`, then a temporary file will be generated and used to dump `spmd.code`.

For Unix-alike systems, the command `cmd <- paste(mpicmd, "-np", nranks, "Rscript", spmd.file, ">", log.file)` is executed via `system(cmd, intern = TRUE, wait = FALSE, ignore.stdout = TRUE, ignore.stderr = TRUE)`. The `log.file` is a temporary file to save the outputs from the `spmd.code`. The results saved to the `log.file` will be read back in and `cat` and return to R.

For Windows, the `cmd` will be `paste(mpicmd, "-np", nranks, "Rscript", spmd.file)` and is executed via `system(cmd, intern = TRUE, wait = FALSE, ignore.stdout = TRUE, ignore.stderr = TRUE)`.

**Value**

Basically, only the PID of the MPI job (in background) will be returned in Linux-alike systems. For Windows, the MPI job is always wait until it is complete.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)> and Drew Schmidt.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

## See Also

`pbdcS::pbdrscript()`.

## Examples

```
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r
```

```
spmd.code <- "
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
allreduce(1)
finalize()
"
pbdMPI::execmpi(spmd.code = spmd.code, nrank = 2L)

spmd.file <- tempfile()
cat("
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
allreduce(2)
finalize()
", file = spmd.file)
pbdMPI::execmpi(spmd.file = spmd.file, nrank = 2L)
```

---

wait

*Wait Functions*

---

## Description

The functions call MPI wait functions.

## Usage

```
wait(request = .pbd_env$SPMD.CT$request,
      status = .pbd_env$SPMD.CT$status)
waitany(count, status = .pbd_env$SPMD.CT$status)
waitsome(count)
waitall(count)
```

**Arguments**

request	a request number.
status	a status number.
count	a count number.

**Details**

These functions are for internal uses. Potentially, they wait after some nonblocking MPI calls.

**Value**

An invisible state of MPI call is returned.

**Author(s)**

Wei-Chen Chen <[wccsnow@gmail.com](mailto:wccsnow@gmail.com)>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: <http://r-pbd.org/>

**Examples**

```
## Not run:
### Save code in a file "demo.r" and run with 2 processors by
### SHELL> mpiexec -np 2 Rscript demo.r

spmd.code <- "
### Initial.
suppressMessages(library(pbdMPI, quietly = TRUE))
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .comm.rank
if(.comm.rank == 0){
  isend(list(x))
}
if(.comm.rank == 1){
  y <- irecv(list(x))
}
wait()
comm.print(y, rank.print = 1L)

### Finish.
finalize()
"
# execmpi(spmd.code, nranks = 2L)
```

*wait*

79

## End(Not run)

# Index

## \*Topic **collective**

- allgather-method, [5](#)
- allreduce-method, [7](#)
- alltoall, [8](#)
- bcast-method, [13](#)
- gather-method, [17](#)
- irecv-method, [50](#)
- isend-method, [53](#)
- recv-method, [56](#)
- reduce-method, [58](#)
- scatter-method, [60](#)
- send-method, [64](#)
- sendrecv-method, [65](#)
- sendrecv.replace-method, [67](#)

## \*Topic **global variables**

- Set global pbd options, [69](#)
- SPMD Control, [72](#)
- SPMD Control Functions, [74](#)

## \*Topic **methods**

- allgather-method, [5](#)
- allreduce-method, [7](#)
- alltoall, [8](#)
- bcast-method, [13](#)
- gather-method, [17](#)
- irecv-method, [50](#)
- isend-method, [53](#)
- recv-method, [56](#)
- reduce-method, [58](#)
- scatter-method, [60](#)
- send-method, [64](#)
- sendrecv-method, [65](#)
- sendrecv.replace-method, [67](#)

## \*Topic **package**

- pbdMPI-package, [3](#)

## \*Topic **programming**

- communicator, [14](#)
- info, [48](#)
- is.comm.null, [51](#)
- MPI array pointers, [54](#)

- probe, [55](#)
- sourcetag, [71](#)
- wait, [77](#)

## \*Topic **utility**

- apply and lapply, [10](#)
- get job id, [19](#)
- global all pairs, [21](#)
- global any and all, [22](#)
- global as.gbd, [24](#)
- global balanc, [26](#)
- global base, [27](#)
- global distance function, [29](#)
- global match.arg, [31](#)
- global pairwise, [32](#)
- global print and cat, [34](#)
- global range, max, and min, [36](#)
- global reading, [37](#)
- global Rprof, [40](#)
- global sort, [41](#)
- global stop and warning, [42](#)
- global timer, [44](#)
- global which, which.max, and which.min, [45](#)
- global writing, [47](#)
- seed for RNG, [62](#)
- Task Pull, [74](#)
- Utility execmpi, [76](#)
- .mpiopt\_init, [70](#)
- .mpiopt\_init (SPMD Control Functions), [74](#)
- .pbd\_env, [70](#), [74](#)
- .pbd\_env (SPMD Control), [72](#)
- allgather, [4](#), [8](#), [9](#), [59](#)
- allgather (allgather-method), [5](#)
- allgather, ANY, ANY, integer-method (allgather-method), [5](#)
- allgather, ANY, missing, integer-method (allgather-method), [5](#)



- allgather, ANY, missing, missing-method (allgather-method), 5
- allgather, integer, integer, integer-method (allgather-method), 5
- allgather, integer, integer, missing-method (allgather-method), 5
- allgather, numeric, numeric, integer-method (allgather-method), 5
- allgather, numeric, numeric, missing-method (allgather-method), 5
- allgather, raw, raw, integer-method (allgather-method), 5
- allgather, raw, raw, missing-method (allgather-method), 5
- allgather-method, 5
- allgatherv, 9
- allgatherv (allgather-method), 5
- allreduce, 4, 6, 19
- allreduce (allreduce-method), 7
- allreduce, ANY, missing-method (allreduce-method), 7
- allreduce, float32, float32-method (allreduce-method), 7
- allreduce, integer, integer-method (allreduce-method), 7
- allreduce, logical, logical-method (allreduce-method), 7
- allreduce, numeric, numeric-method (allreduce-method), 7
- allreduce-method, 7
- alltoall, 8
- anysource (sourcetag), 71
- anytag (sourcetag), 71
- apply and lapply, 10
- arrange.mpi.pts (MPI array pointers), 54
  
- barrier (communicator), 14
- bcast, 4, 61
- bcast (bcast-method), 13
- bcast, ANY-method (bcast-method), 13
- bcast, integer-method (bcast-method), 13
- bcast, numeric-method (bcast-method), 13
- bcast, raw-method (bcast-method), 13
- bcast-method, 13
  
- comm.abort (communicator), 14
- comm.accept (communicator), 14
- comm.all (global any and all), 22
- comm.allcommon (global any and all), 22
- comm.allpairs, 30
- comm.allpairs (global all pairs), 21
- comm.any (global any and all), 22
- comm.as.gbd, 27
- comm.as.gbd (global as.gbd), 24
- comm.balance.info (global balanc), 26
- comm.c2f (communicator), 14
- comm.cat (global print and cat), 34
- comm.connect (communicator), 14
- comm.disconnect (communicator), 14
- comm.dist, 22, 33
- comm.dist (global distance function), 29
- comm.dup (communicator), 14
- comm.end.seed (seed for RNG), 62
- comm.free (communicator), 14
- comm.is.null (communicator), 14
- comm.length (global base), 27
- comm.load.balance, 25, 38, 39, 48
- comm.load.balance (global balanc), 26
- comm.localrank (communicator), 14
- comm.match.arg (global match.arg), 31
- comm.max (global range, max, and min), 36
- comm.mean (global base), 27
- comm.min (global range, max, and min), 36
- comm.pairwise, 30, 33
- comm.pairwise (global pairwise), 32
- comm.print (global print and cat), 34
- comm.range (global range, max, and min), 36
- COMM.RANK, 62
- COMM.RANK (communicator), 14
- comm.rank (communicator), 14
- comm.read.csv (global reading), 37
- comm.read.csv2 (global reading), 37
- comm.read.table, 25–27, 46, 48
- comm.read.table (global reading), 37
- comm.reset.seed (seed for RNG), 62
- comm.Rprof (global Rprof), 40
- comm.sd (global base), 27
- comm.seed.state (seed for RNG), 62
- comm.set.seed (seed for RNG), 62
- COMM.SIZE, 20, 62
- COMM.SIZE (communicator), 14
- comm.size (communicator), 14
- comm.sort (global sort), 41

- comm.split (communicator), 14
- comm.stop (global stop and warning), 42
- comm.stopifnot (global stop and warning), 42
- comm.sum (global base), 27
- comm.timer (global timer), 44
- comm.unload.balance (global balanc), 26
- comm.var (global base), 27
- comm.warning (global stop and warning), 42
- comm.warnings (global stop and warning), 42
- comm.which (global which, which.max, and which.min), 45
- comm.write (global writing), 47
- comm.write.table, 25, 27, 39
- communicator, 14
  
- execmpi (Utility execmpi), 76
  
- finalize, 3
- finalize (communicator), 14
  
- gather, 4, 6, 8, 19, 59
- gather (gather-method), 17
- gather, ANY, ANY, integer-method (gather-method), 17
- gather, ANY, missing, integer-method (gather-method), 17
- gather, ANY, missing, missing-method (gather-method), 17
- gather, integer, integer, integer-method (gather-method), 17
- gather, integer, integer, missing-method (gather-method), 17
- gather, numeric, numeric, integer-method (gather-method), 17
- gather, numeric, numeric, missing-method (gather-method), 17
- gather, raw, raw, integer-method (gather-method), 17
- gather, raw, raw, missing-method (gather-method), 17
- gather-method, 17
- get job id, 19
- get.jid, 75
- get.jid (get job id), 19
- get.sourcetag (sourcetag), 71
- global all pairs, 21
- global any and all, 22
- global as.gbd, 24
- global balanc, 26
- global base, 27
- global distance function, 29
- global match.arg, 31
- global pairwise, 32
- global print and cat, 34
- global range, max, and min, 36
- global reading, 37
- global Rprof, 40
- global sort, 41
- global stop and warning, 42
- global timer, 44
- global which, which.max, and which.min, 45
- global writing, 47
  
- info, 48
- init, 3
- init (communicator), 14
- intercomm.create (communicator), 14
- intercomm.merge (communicator), 14
- iprobe (probe), 55
- irecv, 54, 57, 65
- irecv (irecv-method), 50
- irecv, ANY-method (irecv-method), 50
- irecv, integer-method (irecv-method), 50
- irecv, numeric-method (irecv-method), 50
- irecv, raw-method (irecv-method), 50
- irecv-method, 50
- is.comm.null, 51
- is.finalized (communicator), 14
- isend, 51, 57, 65
- isend (isend-method), 53
- isend, ANY-method (isend-method), 53
- isend, integer-method (isend-method), 53
- isend, numeric-method (isend-method), 53
- isend, raw-method (isend-method), 53
- isend-method, 53
  
- MPI array pointers, 54
  
- pbid\_opt (Set global pbid options), 69
- pbidApply (apply and lapply), 10
- pbidLapply (apply and lapply), 10
- pbidMPI (pbidMPI-package), 3
- pbidMPI-package, 3
- pbidSapply (apply and lapply), 10

- port.close (communicator), 14
- port.open (communicator), 14
- probe, 55
- recv, 51, 54, 65
- recv (recv-method), 56
- recv, ANY-method (recv-method), 56
- recv, integer-method (recv-method), 56
- recv, numeric-method (recv-method), 56
- recv, raw-method (recv-method), 56
- recv-method, 56
- reduce, 4, 6, 8, 19, 59
- reduce (reduce-method), 58
- reduce, ANY, missing-method (reduce-method), 58
- reduce, float32, float32-method (reduce-method), 58
- reduce, integer, integer-method (reduce-method), 58
- reduce, logical, logical-method (reduce-method), 58
- reduce, numeric, numeric-method (reduce-method), 58
- reduce-method, 58
- Rprof, 40
- scatter, 4, 14
- scatter (scatter-method), 60
- scatter, ANY, ANY, integer-method (scatter-method), 60
- scatter, ANY, missing, integer-method (scatter-method), 60
- scatter, ANY, missing, missing-method (scatter-method), 60
- scatter, integer, integer, integer-method (scatter-method), 60
- scatter, integer, integer, missing-method (scatter-method), 60
- scatter, numeric, numeric, integer-method (scatter-method), 60
- scatter, numeric, numeric, missing-method (scatter-method), 60
- scatter, raw, raw, integer-method (scatter-method), 60
- scatter, raw, raw, missing-method (scatter-method), 60
- scatter-method, 60
- seed for RNG, 62
- send, 51, 54, 57
- send (send-method), 64
- send, ANY-method (send-method), 64
- send, integer-method (send-method), 64
- send, numeric-method (send-method), 64
- send, raw-method (send-method), 64
- send-method, 64
- sendrecv, 68
- sendrecv (sendrecv-method), 65
- sendrecv, ANY, ANY-method (sendrecv-method), 65
- sendrecv, integer, integer-method (sendrecv-method), 65
- sendrecv, numeric, numeric-method (sendrecv-method), 65
- sendrecv, raw, raw-method (sendrecv-method), 65
- sendrecv-method, 65
- sendrecv.replace, 66
- sendrecv.replace (sendrecv.replace-method), 67
- sendrecv.replace, ANY-method (sendrecv.replace-method), 67
- sendrecv.replace, integer-method (sendrecv.replace-method), 67
- sendrecv.replace, numeric-method (sendrecv.replace-method), 67
- sendrecv.replace, raw-method (sendrecv.replace-method), 67
- sendrecv.replace-method, 67
- serv.lookup (communicator), 14
- serv.publish (communicator), 14
- serv.unpublish (communicator), 14
- Set global pbd options, 69
- sourcetag, 71
- SPMD Control, 72
- SPMD Control Functions, 74
- spmd.alltoall.double (alltoall), 8
- spmd.alltoall.integer (alltoall), 8
- spmd.alltoall.raw (alltoall), 8
- spmd.alltoallv.double (alltoall), 8
- spmd.alltoallv.integer (alltoall), 8
- spmd.alltoallv.raw (alltoall), 8
- SPMD.CT, 70
- SPMD.CT (SPMD Control Functions), 74
- SPMD.IO, 70
- SPMD.IO (SPMD Control Functions), 74
- SPMD.OP, 70
- SPMD.OP (SPMD Control Functions), 74

SPMD.TP, [70](#)

SPMD.TP (SPMD Control Functions), [74](#)

Task Pull, [74](#)

task.pull, [20](#), [73](#)

task.pull (Task Pull), [74](#)

Utility execmpi, [76](#)

wait, [77](#)

waitall (wait), [77](#)

waitany (wait), [77](#)

waitsome (wait), [77](#)