Package ‘pbdMPI’

December 8, 2019

Version 0.4-2
Date 2019-12-01
Title Programming with Big Data -- Interface to MPI
Depends R (>= 3.5.0), methods
Imports rlecuyer, float
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 topics documented:

- **R Core team [ctb]** (some functions are modified from the base packages),
- **Sebastien Lamy de la Chapelle [aut]** (fix check type for send/recv long vectors)

**Repository** CRAN

**Date/Publication** 2019-12-08 10:30:12 UTC

---

<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbdMPI-package</td>
<td>3</td>
</tr>
<tr>
<td>allgather-method</td>
<td>5</td>
</tr>
<tr>
<td>allreduce-method</td>
<td>7</td>
</tr>
<tr>
<td>alltoall</td>
<td>8</td>
</tr>
<tr>
<td>apply and lapply</td>
<td>10</td>
</tr>
<tr>
<td>bcast-method</td>
<td>13</td>
</tr>
<tr>
<td>communicator</td>
<td>14</td>
</tr>
<tr>
<td>gather-method</td>
<td>17</td>
</tr>
<tr>
<td>Get Configures Used at Compiling Time</td>
<td>19</td>
</tr>
<tr>
<td>get job id</td>
<td>21</td>
</tr>
<tr>
<td>global all pairs</td>
<td>22</td>
</tr>
<tr>
<td>global any and all</td>
<td>24</td>
</tr>
<tr>
<td>global as.gbd</td>
<td>25</td>
</tr>
<tr>
<td>global balance</td>
<td>27</td>
</tr>
<tr>
<td>global base</td>
<td>29</td>
</tr>
<tr>
<td>global distance function</td>
<td>30</td>
</tr>
<tr>
<td>global match.arg</td>
<td>32</td>
</tr>
<tr>
<td>global pairwise</td>
<td>33</td>
</tr>
<tr>
<td>global print and cat</td>
<td>35</td>
</tr>
<tr>
<td>global range, max, and min</td>
<td>37</td>
</tr>
<tr>
<td>global reading</td>
<td>38</td>
</tr>
<tr>
<td>global Rprof</td>
<td>41</td>
</tr>
<tr>
<td>global sort</td>
<td>42</td>
</tr>
<tr>
<td>global stop and warning</td>
<td>43</td>
</tr>
<tr>
<td>global timer</td>
<td>45</td>
</tr>
<tr>
<td>global which, which.max, and which.min</td>
<td>46</td>
</tr>
<tr>
<td>global writing</td>
<td>48</td>
</tr>
<tr>
<td>info</td>
<td>49</td>
</tr>
<tr>
<td>irecv-method</td>
<td>51</td>
</tr>
<tr>
<td>is.comm.null</td>
<td>52</td>
</tr>
<tr>
<td>isend-method</td>
<td>54</td>
</tr>
<tr>
<td>MPI array pointers</td>
<td>55</td>
</tr>
<tr>
<td>Package Tools</td>
<td>56</td>
</tr>
<tr>
<td>probe</td>
<td>58</td>
</tr>
<tr>
<td>recv-method</td>
<td>59</td>
</tr>
<tr>
<td>reduce-method</td>
<td>61</td>
</tr>
<tr>
<td>scatter-method</td>
<td>62</td>
</tr>
<tr>
<td>seed for RNG</td>
<td>64</td>
</tr>
<tr>
<td>send-method</td>
<td>66</td>
</tr>
</tbody>
</table>
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 zxf pbdMPI_0.1-0.tar.gz
> R CMD INSTALL pbdMPI
```

Other arguments include

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>--with-mpi-type</td>
<td>OPENMPI</td>
</tr>
<tr>
<td>--with-mpi-include</td>
<td>${MPI_ROOT}/include</td>
</tr>
<tr>
<td>--with-mpi-libpath</td>
<td>${MPI_ROOT}/lib</td>
</tr>
<tr>
<td>--with-mpi</td>
<td>${MPI_ROOT}</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbdMPI/inst/examples/test_spmd/</td>
<td>major SPMD functions</td>
</tr>
<tr>
<td>pbdMPI/inst/examples/test_rmpi/</td>
<td>analog to <code>Rmpi</code></td>
</tr>
<tr>
<td>pbdMPI/inst/examples/test_parallel/</td>
<td>analog to <code>parallel</code></td>
</tr>
<tr>
<td>pbdMPI/inst/examples/test_performance/</td>
<td>performance tests</td>
</tr>
<tr>
<td>pbdMPI/inst/examples/test_s4/</td>
<td>S4 extension</td>
</tr>
<tr>
<td>pbdMPI/inst/examples/test_cs/</td>
<td>client/server examples</td>
</tr>
<tr>
<td>pbdMPI/inst/examples/test_long_vector/</td>
<td>long vector examples</td>
</tr>
</tbody>
</table>

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/](http://r-pbd.org/)

**See Also**

`allgather()`, `allreduce()`, `bcast()`, `gather()`, `reduce()`, `scatter()`.

**Examples**

```R
## 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)"
```
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

x an object to be gathered from all ranks.

x.buffer a buffer to hold the return object which probably has `size of x' times `comm.size(comm)' with the same type of x.

x.count a vector of length `comm size' containing all object lengths.

displs c(0L, cumsum(x.count)) by default.

comm a communicator number.

unlist 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.allgatherv.*() 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.allgatherv.*`:  

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/](http://r-pbd.org/)

See Also

gather(), allreduce(), reduce().

Examples

```r
### 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)
```
### 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**

```r
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.*`:

```r
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

```r
### 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 <- 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()
```

```
```
Description

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

Usage

```r
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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

References

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)
apply and lapply

Parallel Apply and Lapply Functions

Description

The functions are parallel versions of apply and lapply functions.

Usage

```r
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,
```
apply and lapply

```r
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 `pbdApply()` or a list in `pbdLapply()` and `pbdSapply()`.
- **MARGIN**: MARGIN as in the `apply()`.
- **FUN**: as in the `apply()`.
- **...**: optional arguments to `FUN`.
- **simplify**: as in the `sapply()`.
- **USE.NAMES**: as in the `sapply()`.
- **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.mode = "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 exists 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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.
References

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

Examples

```r
### 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

```r
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

```r
### 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, nranks = 2L)
```

---

**communicator**

**Communicator Functions**

**Description**

The functions provide controls to communicators.

**Usage**

```r
comm.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)
```
communicator

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

comm a communicator number.
mpi.finalize if MPI should be shutdown.
set.seed if a random seed preset.
port.name a port name with default maximum length 1024 characters for OpenMPI.
info a info number.
rank.root a root rank.
newcomm a new communicator number.
color, key, serv.name, errorcode, intercomm, high, local.comm, local.leader, peer.comm, remote.leader, newintercomm, tag

control of subset assignment.
control of rank assignment.
a service name.
an error code to abort MPI.
a intercommunicator number.
used to order the groups within comm.
a local communicator number.
the leader number of local communicator.
a peer communicator number.
the remote leader number of peer communicator.
a new intercommunicator number.
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.
**gather-method**

**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/](http://r-pbd.org/)

**Examples**

```r
## 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()
.com.size <- comm.size()
.com.rank <- comm.rank()

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

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

---

**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

- **x**: an object to be gathered from all ranks.
- **x.buffer**: a buffer to hold the return object which probably has ‘size of x’ times ‘comm size’ with the same type of x.
- **x.count**: a vector of length ‘comm size’ containing all object lengths.
- **displs**: `c(0L,cumsum(x.count))` by default.
- **rank.dest**: a rank of destination where all x gather to.
- **comm**: a communicator number.
- **unlist**: 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")
Get Configures Used at Compiling Time

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

```r
### 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 <- gather(matrix(x, nrow = 1))
comm.print(y)
y <- gather(x, double(N * .comm.size))
comm.print(y)

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

Get Configures Used at Compiling Time

Functions to Get MPI and/or pbdMPI Configures Used at Compiling Time

Description
These functions are designed to get MPI and/or pbdMPI configures that were usually needed at the time of pbdMPI installation. In particular, to configure, link, and compile with ‘libmpi*.so’ or so.
Get Configures Used at Compiling Time

Usage

get.conf(arg, arch = '\'', package = "pbdMPI", return = FALSE)
get.lib(arg, arch, package = "pbdPROF")
get.sysenv(flag)

Arguments

arg an argument to be searched in the configuration file
arch system architecture
package package name
return to return (or print if FALSE) the search results or not
flag a system flag that is typically used in windows environment set.

Details

get.conf() and get.lib() are typically used by ‘pbd*/configure.ac’, ‘pbd*/src/Makevars.in’, and/or ’pbd*/src/Makevar.win’ to find the default configurations from 'pbd*/etc$R_ARCH/Makconf'.
get.sysenv() is only called by ‘pbdMPI/src/Makevars.win’ to obtain possible MPI dynamic/static library from the environment variable ‘MPI_ROOT’ preset by users.

Value

Typically, there are no return values, but the values are cat() to scrrn or stdin.

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

```r
## Not run:
library(pbdMPI)
if(Sys.info()$"sysname" != "Windows"){
  get.conf("MPI_INCLUDE_PATH"); cat("\n")
  get.conf("MPI_LIBPATH"); cat("\n")
  get.conf("MPI_LIBNAME"); cat("\n")
  get.conf("MPI_LIBS"); cat("\n")
} else{
  get.conf("MPI_INCLUDE", "/i386"); cat("\n")
  get.conf("MPI_LIB", "/i386"); cat("\n")
}
```
Description

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

Usage

```r
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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.
### 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.
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)
Usage

\[
\text{comm.allpairs}(N, \text{diag} = \text{FALSE}, \text{symmetric} = \text{TRUE}, \\
\quad \text{comm} = \text{.pbd_env$SPMD.CT$comm})
\]

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 \text{diag} and \text{symmetric}. If \text{diag} = \text{TRUE} and \text{symmetric} = \text{FALSE}, then this case has the maximum number of rows, \(N^2\).

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/](http://r-pbd.org/)

See Also

\text{comm.dist}().

Examples

```r
## 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**

- **x**: a vector.
- **na.rm**: if NA removed or not.
- **comm**: a communicator number.
- **lazy.check**: if TRUE, then `allreduce` is used to check all ranks, otherwise, `allgather` 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 scaler.  
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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.
global as.gbd

References

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

Examples

```r
## 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(spmd.code, nranks = 2L)

## End(Not run)
```

---

global as.gbd | Global As GBD Function

## Description

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

\[
\text{comm.as.gbd}(X, \text{balance.method} = \text{.pbd.env$SPMD.IO$balance.method}, \text{rank.source} = \text{.pbd.env$SPMD.CT$rank.source}, \text{comm} = \text{.pbd.env$SPMD.CT$comm})
\]

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/](http://r-pbd.org/)

See Also

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

Examples

```r
### 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)
```

global balance

comm.print(X.gbd, all.rank = TRUE)

### Finish.
finalize()

pbdMPI::execmpi(spmd.code, n ranks = 4L)

---

### Global Balance Functions

**Description**

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

**Usage**

```r
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**

- `X.gbd`: a gbd data.frame (or matrix).
- `bal.info`: a balance information returned from `comm.balance.info()`. If `NULL`, then this will be generated inside `comm.load.balance()`.
- `new.X.gbd`: a new gbd of `X.gbd` (may be generated from `comm.load.balance()`).
- `comm`: 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/](http://r-pbd.org/)

See Also

`comm.read.table()`, `comm.write.table()`, and `comm.as.gbd()`.

Examples

```r
## 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)
```
Global Base Functions

Description

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

Usage

```r
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/](http://r-pbd.org/)

Examples

```r
## 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("2 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, nranks = 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

- **X.gbd**: a gbd matrix.
- **method**: as in `dist()`.
- **diag**: as in `dist()`.
- **upper**: as in `dist()`.
- **p**: as in `dist()`.
global distance function

comm a communicator number.
return.type 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 $N \times N$ 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>, 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

```r
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

- `arg`, `choices`, `several.ok`  
  see `match.arg()`
- `...`  
  ignored.
- `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).
- `mpi.finalize`  
  if MPI should be shutdown.
- `quit`  
  if quit R when errors happen.
**Author(s)**

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

**References**

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

---

### Description

This function provides global pairwise evaluations.

### Usage

```r
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

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

### Details

This function evaluates the objective function \( \text{FUN}(X[i,], X[j,]) \) (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 \(\text{pairid.gbd}\) is provided, e.g. from \text{comm.pairwise}(). i.e. \(X\) is exactly the same in every ranks, but \(\text{pairid.gbd}\) is different and in gbd format indicating the row pair \((i,j)\) should be evaluated. The returning gbd matrix is ordered and indexed by \(\text{pairid.gbd}\).

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 \(\text{pairid.gbd}\) 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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

References

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

See Also

comm.pairwise(), and comm.dist().

Examples

```r
## 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 = 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("c", allgather(dist.common[, 3]))
d.2 <- do.call("c", allgather(dist.gbd[, 3]))
```
global print and cat

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

### Finish.
finalize()
```

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

## End(Not run)

---

### 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**

```r
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.
global print and cat

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>, 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, nranks = 2L)

## End(Not run)
Global Range, Max, and Min Functions

Description

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

Usage

```r
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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

References

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

Examples

```r
## 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.
 suppressWarnings(library(pbdMPI, quietly = TRUE))
 init()
```

if(comm.size() != 2){
    comm.cat("2 processors are required.
", 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 Functions

#### Description

These functions are global reading from specified file.

#### Usage

```r
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)
```

```r
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.csv2().
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 disable 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

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

#### Check.
if(comm.size() != 4){
  comm.stop("4 processors are requried.")
}

#### 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()
global Rprof

```
# execmpi(spmd.code, nranks = 4L)
## End(Not run)
```

---

**Description**

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

**Usage**

```r
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 `Rprof()`.
- `append` as in `Rprof()`.
- `interval` as in `Rprof()`.
- `memory.profiling` as in `Rprof()`.
- `gc.profiling` as in `Rprof()`.
- `line.profiling` as in `Rprof()`.
- `numfiles` as in `Rprof()`.
- `bufsize` as in `Rprof()`.
- `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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.
global sort

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 function.

**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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: http://r-pbd.org/
### Examples

```r
# 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
... variables to be cat.
call. see stop() and warnings().
immediate. see stop() and warnings().
domain see stop() and warnings().
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).
mpi.finalize if MPI should be shutdown.
quit 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.
global timer

A Timing Function for SPMD Routines

Description

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

Usage

```r
comm.timer(timed, comm = .pbd_env$SPMD.CT$comm)
```
global which, which.max, and which.min

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

```r
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.
**global which, which.max, and which.min**

**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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

**See Also**

comm.read.table()

**Examples**

```r
## 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("2 processors are requiried.
", 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, nranks = 2L)
## End(Not run)
```
Global Writing Functions

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

```r
## 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("2 processors are required.
", quiet = TRUE)
  finalize()
}

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

### Finish.
finalize()
"
# execmpi(spmd.code, n ranks = 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>, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

References

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

Examples

```r
## 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()
```
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

```r
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

- `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.
- `request`: a request number.
- `status`: a status number.

Details

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

**Warning:** `irecv()` is not safe for \( \text{R} \) since \( \text{R} \) 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 `spmd.irecv.*()`:

- `signature(x = "ANY")`
- `signature(x = "integer")`
- `signature(x = "numeric")`
- `signature(x = "raw")`
is.comm.null

Check if a MPI_COMM_NULL

Description

The functions check MPI_COMM_NULL.

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()
.com.size <- comm.size()
.com.rank <- comm.rank()

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

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

## End(Not run)
is.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

```r
## 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, nranks = 2L)
## End(Not run)
```
**isend-method**

A Rank Send (Nonblocking) an Object to the Other Rank

**Description**

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

**Usage**

```r
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,
       check.type = .pbd_env$SPMD.CT$check.type)
```

**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.
- `check.type` if checking data type first for handshaking.

**Details**

A corresponding `recv()` or `irecv()` should be evoked at the corresponding rank `rank.dest`. See details of `send()` for the arguments `check.type`.

**Value**

A NULL is returned by default.

**Methods**

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

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

\texttt{arrange.mpi.apts()}

Details

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

If the package loads first, then this sets '\texttt{.__MPI_APTS__}' in the .\texttt{GlobalEnv} of \texttt{R}. If the package does not load before other MPI APIs, then this points an structure point to the external memory according to '\texttt{.__MPI_APTS__}', i.e. pre-set by other MPI APIs.

\texttt{pbdMPI/R/arrange.mpi.apts} provides the \texttt{R} code, and \texttt{pbdMPI/src/pkg_*.*} provides the details of this call.

Value

'\texttt{.__MPI_APTS__}' is set in the .\texttt{GlobalEnv} of \texttt{R}.

Author(s)

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

References

Programming with Big Data in R Website: \url{http://r-pbd.org/}

Examples

\begin{verbatim}
## Not run:
### See source code for the details.
## End(Not run)
\end{verbatim}

---

Description

These functions are designed to get or print MPI_COMM pointer and its address when the SPMD code in \texttt{R} be a foreign application of other applications.

Usage

\begin{verbatim}
get.mpi.comm.ptr(comm = .\texttt{pdb_env$SPMD.CT$comm}, show.msg = FALSE)
addr.mpi.comm.ptr(comm.ptr)
\end{verbatim}
Arguments

comm          a communicator number.
comm.ptr      a communicator pointer.
show.msg      if showing message for debug only.

Details

get.mpi.comm.ptr() returns an R external pointer that points to the address of the comm.
addr.mpi.comm.ptr() takes the R external points, and prints the address of the comm. This function is mainly for debugging.

Value

get.mpi.comm.ptr() returns an R external pointer.
addr.mpi.comm.ptr() prints the comm pointer address and the address of MPI_COMM_WORLD.

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

### 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()

ptr1 <- get.mpi.comm.ptr(1, show.msg = TRUE)
addr.mpi.comm.ptr(ptr1)

comm.split(color = as.integer(comm.rank()/2), key = comm.rank())

ptr1.new <- get.mpi.comm.ptr(1, show.msg = TRUE)
addr.mpi.comm.ptr(ptr1.new)

### Finish.
finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks = 4L)
**Probe Functions**

**Description**

The functions call MPI probe functions.

**Usage**

```r
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)
```

```r
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**

- `rank.source`: a source rank where an object sent from.
- `tag`: a tag number.
- `comm`: a communicator number.
- `status`: 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/](http://r-pbd.org/)

**Examples**

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

## End(Not run)```
recv-method

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,
     check.type = .pbd_env$SPMD.CT$check.type)

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.
check.type if checking data type first for handshaking.

Details

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

These are high level S4 methods. By default, check.type is TRUE and an additional send()/recv() will make a handshaking call first, then deliver the data next. i.e. an integer vector of length two (type and length) will be deliver first between send() and recv() to ensure a buffer (of right type and right size/length) is properly allocated at the rank.dest side.

Currently, four data types are considered: integer, double, raw/byte, and default/raw.object. The default method will make a serialize() call first to convert the general R object into a raw vector before sending it away. After the raw vector is received at the rank.dest side, the vector will be unserialize() back to the R object format.

check.type set as FALSE will stop the additional handshaking call, but the buffer should be prepared carefully by the user self. This is typically for the advanced users and more specifically calls are needed. i.e. calling those spmd.send.integer with spmd.recv.integer correspondingly.

check.type also needs to be set as FALSE for more efficient calls such as isend()/recv() or send()/irecv(). Currently, no check types are implemented in those mixed calls.

Value

An object is returned by default and the buffer will be overwritten implicitly.
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>, 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(spmd.code, nranks = 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

```r
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

- `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 on combine all `x`.
- `rank.dest`: a rank of destination where all `x` reduce to.
- `comm`: 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.*`:

```r
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")
```
scatter-method

**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/](http://r-pbd.org/)

**See Also**

allgather(), gather(), reduce().

**Examples**

```r
### 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(spmd.code = spmd.code, nranks = 2L)
```

---

**scatter-method** A Rank Scatter Objects to Every Rank
scatter-method

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

```r
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

- `x`: an object of length ‘comm size’ to be scattered to all ranks.
- `x.buffer`: a buffer to hold the return object which probably has 'size of element of `x`' with the same type of the element of `x`.
- `x.count`: a vector of length ‘comm size’ containing all object lengths.
- `displs`: `c(0L, cumsum(x.count))` by default.
- `rank.source`: a rank of source where elements of `x` scatter from.
- `comm`: 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 `spmd.scatterv.*()` is called.

Value

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

Methods

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

```r
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.*()`:

```r
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

bcast().

Examples

```r
### 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, nranks = 2L)
```

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 rlecuyer.
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 every- 
thing 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

Number Package with Many Long Streams and Substreams. Operations Research, 50(6), 1073-
1075.
http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf

age, URL https://cran.r-project.org/package=rlecuyer

Programming with Big Data in R Website: http://r-pbd.org/
See Also

.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

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

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

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

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

### Finish.
finalize()
"

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

## End(Not run)

__send-method__  

**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,
     check.type = .pbd_env$SPMD.CT$check.type)

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.
check.type if checking data type first for handshaking.

Details

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

These are high level S4 methods. By default, check.type is TRUE and an additional send()/recv() will make a handshaking call first, then deliver the data next. i.e. an integer vector of length two (type and length) will be deliver first between send() and recv() to ensure a buffer (of right type and right size/length) is properly allocated at the rank.dest side.

Currently, four data types are considered: integer, double, raw/byte, and default/raw.object. The default method will make a serialize() call first to convert the general R object into a raw vector before sending it away. After the raw vector is received at the rank.dest side, the vector will be unserialize() back to the R object format.

check.type set as FALSE will stop the additional handshaking call, but the buffer should be prepared carefully by the user self. This is typically for the advanced users and more specifically calls are needed. i.e. calling those spmd.send.integer with spmd.recv.integer correspondingly.

check.type also needs to be set as FALSE for more efficient calls such as isend()/recv() or send()/irecv(). Currently, no check types are implemented in those mixed calls.

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

issent(), 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

  x                  an object to be sent from a rank.
  x.buffer           a buffer to store x sent from the other rank.
  rank.dest          a rank of destination where x send to.
  send.tag           a send tag number.
  rank.source        a source rank where x sent from.
  recv.tag           a receive tag number.
  comm               a communicator number.
  status             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>, 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()
.com.size <- comm.size()
.com.rank <- comm.rank()

### Examples.
N <- 5
x <- (1:N) + N * .com.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) %%
```

sendrecv.replace-method

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.
rank.dest a rank of destination where x send to.
send.tag a send tag number.
rank.source a source rank where x sent from.
recv.tag a receive tag number.
comm a communicator number.
status 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>, 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, nranks = 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.
Set global pbd options

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> 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)
**Description**

The functions extract MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_status.source and MPI\_status.tag.

**Usage**

```r
anysource()
anntag()
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>, George Ostrouplov, Drew Schmidt, Pragneshkumar Patel, and Hao Yu.

**References**

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

**Examples**

```r
## 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()
.com.size <- comm.size()
.com.rank <- comm.rank()
if(.com.size < 2)
  comm.stop("At least two processors are required.")
```
### 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 = anysource(), 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:

<table>
<thead>
<tr>
<th>Elements</th>
<th>Default</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>comm</td>
<td>0L</td>
<td>a communicator index</td>
</tr>
<tr>
<td>intercomm</td>
<td>2L</td>
<td>an inter communicator index</td>
</tr>
<tr>
<td>info</td>
<td>0L</td>
<td>an info index</td>
</tr>
<tr>
<td>newcomm</td>
<td>1L</td>
<td>a new communicator index</td>
</tr>
<tr>
<td>op</td>
<td>&quot;sum&quot;</td>
<td>an operation</td>
</tr>
<tr>
<td>port.name</td>
<td>&quot;spmdport&quot;</td>
<td>an operation</td>
</tr>
<tr>
<td>print.all.rank</td>
<td>FALSE</td>
<td>if all ranks print message</td>
</tr>
<tr>
<td>print.quiet</td>
<td>FALSE</td>
<td>if print/cat rank information</td>
</tr>
<tr>
<td>rank.root</td>
<td>0L</td>
<td>a rank of root</td>
</tr>
<tr>
<td>rank.source</td>
<td>0L</td>
<td>a rank of source</td>
</tr>
<tr>
<td>rank.dest</td>
<td>1L</td>
<td>a rank of destination</td>
</tr>
<tr>
<td>request</td>
<td>0L</td>
<td>a request index</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Elements</th>
<th>Default</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>max.read.size</td>
<td>5.2e6</td>
<td>max of reading size (5 MB)</td>
</tr>
<tr>
<td>max.test.lines</td>
<td>500</td>
<td>max of testing lines</td>
</tr>
<tr>
<td>read.method</td>
<td>&quot;gbd&quot;</td>
<td>default reading method</td>
</tr>
<tr>
<td>balance.method</td>
<td>&quot;block&quot;</td>
<td>default load balance method</td>
</tr>
</tbody>
</table>

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:

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

See `task.pull()` for details.

**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/](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. The values are not supposed to be changed in general.

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

```r
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)
```
Task Pull

Arguments

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

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>, 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

```r
## 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)", nranks = 4L)

## End(Not run)
```
Utility execmpi

_c Execute MPI code in system_

**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 spmd.file").

**Usage**

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

**Arguments**

- `spmd.code` SPMD code to be run via mpicmd and Rscript.
- `spmd.file` a file contains SPMD code to be run via mpicmd and Rscript.
- `mpicmd` MPI executable command. If NULL, system default will be searched.
- `nranks` number of processes to run the SPMD code envoked by mpicmd.
- `verbose` 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," 2>&1 & echo \"PID=$!\" ")` 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> and Drew Schmidt.

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

See Also
pbdCS::pbdRscript().

Examples

```r
### 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, nranks = 2L)

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

---

**wait**

**Wait Functions**

**Description**

The functions call MPI wait functions.

**Usage**

```r
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)
```
wait

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>, 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)
## End (Not run)
## Index

**Topic collective**
- allgather-method, 5
- allreduce-method, 7
- alltoall, 8
- bcast-method, 13
- gather-method, 17
- irecv-method, 51
- isend-method, 54
- recv-method, 59
- reduce-method, 61
- scatter-method, 62
- send-method, 66
- sendrecv-method, 68
- sendrecv.replace-method, 70

**Topic global variables**
- Set global pbd options, 72
- SPMD Control, 75
- SPMD Control Functions, 77

**Topic methods**
- allgather-method, 5
- allreduce-method, 7
- alltoall, 8
- bcast-method, 13
- gather-method, 17
- irecv-method, 51
- isend-method, 54
- recv-method, 59
- reduce-method, 61
- scatter-method, 62
- send-method, 66
- sendrecv-method, 68
- sendrecv.replace-method, 70

**Topic package**
- pbdMPI-package, 3

**Topic programming**
- communicator, 14
- info, 49
- is.comm.null, 52
- MPI array pointers, 55

- probe, 58
- sourcetag, 74
- wait, 80

**Topic utility**
- apply and lapply, 10
- Get Configures Used at Compiling Time, 19
- get job id, 21
- global all pairs, 22
- global any and all, 24
- global as.gbd, 25
- global balanc, 27
- global base, 29
- global distance function, 30
- global match.arg, 32
- global pairwise, 33
- global print and cat, 35
- global range, max, and min, 37
- global reading, 38
- global Rprof, 41
- global sort, 42
- global stop and warning, 43
- global timer, 45
- global which, which.max, and which.min, 46
- global writing, 48
- Package Tools, 56
- seed for RNG, 64
- Task Pull, 77
- Utility execmpi, 79
- .mpiopt_init, 73
- .mpiopt_init (SPMD Control Functions), 77
- .pbd_env, 73, 77
- .pbd_env (SPMD Control), 75

- addr.mpi.comm.ptr (Package Tools), 56
- allgather, 4, 8, 10, 62
- allgather(allgather-method), 5
COMM.SIZE (communicator), 14
comm.size (communicator), 14
comm.sort (global sort), 42
comm.split (communicator), 14
comm.stop (global stop and warning), 43
comm.stopifnot (global stop and warning), 43
comm.sum (global base), 29
comm.timer (global timer), 45
comm.unload.balance (global balance), 27
comm.var (global base), 29
comm.warning (global stop and warning), 43
comm.warnings (global stop and warning), 43
comm.which (global which, which.max, and which.min), 46
comm.write (global writing), 48
comm.write.table, 26, 28, 40
communicator, 14
execmpi (Utility execmpi), 79
finalize, 4
finalize (communicator), 14
gather, 4, 6, 8, 19, 62
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-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 Configures Used at Compiling Time, 19
get job id, 21
get.conf (Get Configures Used at Compiling Time), 19
get.jid, 78
get.jid (get job id), 21
get.lib (Get Configures Used at Compiling Time), 19
get.mpi.comm.ptr (Package Tools), 56
get.sourcetag (sourcetag), 74
get.sysenv (Get Configures Used at Compiling Time), 19
global all pairs, 22
global any and all, 24
global as.gbd, 25
global base, 27
global distance function, 30
global match.arg, 32
global pairwise, 33
global print and cat, 35
global range, max, and min, 37
global reading, 38
global Rprof, 41
global sort, 42
global stop and warning, 43
global timer, 45
global which, which.max, and which.min, 46
global writing, 48
info, 49
init, 4
init (communicator), 14
intercomm.create (communicator), 14
intercomm.merge (communicator), 14
iprobe (probe), 58
irecv, 55, 60, 68
irecv (irecv-method), 51
irecv, ANY-method (irecv-method), 51
irecv, integer-method (irecv-method), 51
irecv, numeric-method (irecv-method), 51
irecv, raw-method (irecv-method), 51
irecv-method, 51
is.comm.null, 52
is.finalized (communicator), 14
isend, 52, 60, 68
isend (isend-method), 54
isend, ANY-method (isend-method), 54
isend, integer-method (isend-method), 54
isend, numeric-method (isend-method), 54
isend, raw-method (isend-method), 54
isend-method, 54

MPI array pointers, 55

Package Tools, 56
pbd_opt (Set global pbd options), 72
pbdApply (apply and lapply), 10
pbdLapply (apply and lapply), 10
pbdMPI (pbdMPI-package), 3
pbdMPI-package, 3
pbdSapply (apply and lapply), 10

port.close (communicator), 14
port.open (communicator), 14
probe, 58
recv, 52, 55, 68
recv (recv-method), 59
recv, ANY-method (recv-method), 59
recv, integer-method (recv-method), 59
recv, numeric-method (recv-method), 59
recv, raw-method (recv-method), 59
recv-method, 59
reduce, 4, 6, 8, 19, 62
reduce (reduce-method), 61
reduce, ANY, missing-method (reduce-method), 61
reduce, float32, float32-method (reduce-method), 61
reduce, integer, integer-method (reduce-method), 61
reduce, logical, logical-method (reduce-method), 61
reduce, numeric, numeric-method (reduce-method), 61
reduce-method, 61
Rprof, 41
runmpi (Utility execmpi), 79

scatter, 4, 14
scatter (scatter-method), 62
scatter, ANY, ANY, integer-method (scatter-method), 62
scatter, ANY, missing, integer-method (scatter-method), 62
scatter, ANY, missing, missing-method (scatter-method), 62
scatter, integer, integer-method (scatter-method), 62
scatter, integer, integer, missing-method (scatter-method), 62
scatter, numeric, numeric, integer-method (scatter-method), 62
scatter, numeric, numeric, missing-method (scatter-method), 62
scatter, raw, raw, integer-method (scatter-method), 62
scatter, raw, raw, missing-method (scatter-method), 62
scatter-method, 62
seed for RNG, 64
send, 52, 55, 60
send (send-method), 66
send, ANY-method (send-method), 66
send, integer-method (send-method), 66
send, numeric-method (send-method), 66
send, raw-method (send-method), 66
send-method, 66
sendrecv, 72
sendrecv (sendrecv-method), 68
sendrecv, ANY, ANY-method (sendrecv-method), 68
sendrecv, integer, integer-method (sendrecv-method), 68
sendrecv, numeric, numeric-method (sendrecv-method), 68
sendrecv, raw, raw-method (sendrecv-method), 68
sendrecv-method, 68
sendrecv.replace, 70
sendrecv.replace (sendrecv.replace-method), 70
sendrecv.replace, ANY-method (sendrecv.replace-method), 70
sendrecv.replace, integer-method (sendrecv.replace-method), 70
sendrecv.replace, numeric-method (sendrecv.replace-method), 70
sendrecv.replace-method, 70
serv.lookup (communicator), 14
serv.publish (communicator), 14
serv.unpublish (communicator), 14
Set global pbd options, 72
sourcetag, 74
SPMD Control, 75
SPMD Control Functions, 77
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, 73
SPMD.CT (SPMD Control Functions), 77
SPMD.DT (SPMD Control Functions), 77
SPMD.IO, 73
SPMD.IO (SPMD Control Functions), 77
SPMD.OP, 73
SPMD.OP (SPMD Control Functions), 77
SPMD.TP, 73
SPMD.TP (SPMD Control Functions), 77

Task Pull, 77
task.pull, 22, 76
task.pull (Task Pull), 77

Utility execmpi, 79

wait, 80
waitall (wait), 80
waitany (wait), 80
waitsome (wait), 80