Package ‘Rdsm’
February 19, 2015

Version 2.1.1

Author  Norm Matloff <normmatloff@gmail.com>
Maintainer  Norm Matloff <normmatloff@gmail.com>

Date  10/01/2014

Title  Threads Environment for R

Description  Provides a threads-type programming environment for R.
The package gives the R programmer the clearer, more concise
shared memory world view, and in some cases gives superior
performance as well. In addition, it enables parallel processing on
very large, out-of-core matrices.

OS_type  unix
Imports  bigmemory (>= 4.0.0), parallel
Suggests  synchronicity

LazyLoad  no
License  GPL (>= 2)
Repository  CRAN
NeedsCompilation  no
Date/Publication  2014-10-08 05:51:02

R topics documented:

Rdsm-package .................................................. 2
barr ......................................................... 4
getidxs .................................................... 4
getmatrix .................................................. 5
loadex ..................................................... 6
makebarr ................................................... 7
mgrinit ....................................................... 7
mgrmakelock ................................................ 8
mgrmakevar ................................................ 9
rdsmlock .................................................... 9
readsync .................................................. 11
stoprdsm .................................................. 12
Rdsm-package

Index

Rdsm-package

Adds a threaded parallel programming paradigm to R.

Description

This package provides a parallel shared-memory programming paradigm for R, very similar to threaded programming in C/C++. This enables the programmer to write simpler, clearer code. Furthermore, in some applications this package produces significantly faster code, compared to versions written for other parallel R libraries. It also allows placing very large matrices in secondary storage, while treating them as being in shared memory.

Details

Package: Rdsm
Type: Package
Version: 2.1.1
Date: 2014-02-16
License: GPL (>= 2)

List of functions:

initialization, run at manager:

mgrinit(): initialize system
mgrmakevar(): create a shared variable
mgrmakelock(): create a lock
makebarr(): create a barrier

called by applications:

barr(): barrier call
rdsmlock(): lock operation (via realrdsmlock())
rdsmunlock(): unlock operation (via realrdsmunlock())

application utilities:

getidxs(): partition a set of indices for work assignment
getmatrix(): allow a matrix to be referenced regardless of whether it is specified as a bigmemory object, a bigmemory descriptor, or via a quoted name
stoprdsm(): shut down cluster and clean up files

Built-in variables accessible by the threads, at the worker nodes:
To run, set up a cluster via the `parallel` package; we’ll refer to the R process from which this is done as the manager; the processes running in the cluster will be called workers. Create the application’s shared variables from the manager, using `mgrmakevar()`. Launch the worker threads, again from the manager, by the `parallel` call `clusterEvalQ()` or `clusterCall()`. One typically codes so that the results are in shared variables. See examples below, and more in the examples/ directory in this distribution.

The shared variables are read to/written by any of the workers and the manager. In fact, while an Rdsm application is running, other R processes on the same machine (or a different machine sharing the same file system, if the variables are filebacked) can access the shared variables. See the file `ExternalAccess.txt` in the doc/ directory.

Rdsm uses the `bigmemory` library to store its shared variables. Though the latter can work on a (physical) cluster of several machines sharing a file system, Rdsm does not run on such systems at this time.

Further documentation in the doc/ directory.

Author(s)

Norm Matloff <matloff@cs.ucdavis.edu>

See Also

`mgrinit`, `mgrmakevar`, `mgrmakelock`, `barr`, `rdsmlock`, `rdsmunlock`, `getidxs`, `getmatrix`

Examples

```r
library(parallel)
c2 <- makeCluster(2)  # form 2-thread Snow cluster
mgrinit(c2)  # initialize Rdsm
gmrmakevar(c2,"m",2,2)  # make a 2x2 shared matrix
m[,] <- 3  # 2x2 matrix of all 3s  
# example of shared memory:
# at each thread, set id to Rdsm built-in ID variable for that thread
clusterEvalQ(c2,id <- myinfo$id)
clusterEvalQ(c2,m[1,1] <- id^2)  # assignment executed by each thread
m[,]  # top row of m should now be (1,4)

# matrix multiplication; the product u %*% v is computed, product
# placed in w

# note again:  mmul() call will be executed by each thread

mmul <- function(u,v,w) {
    require(parallel)
    # decide which rows of u this thread will work on
    myidxs <- splitIndices(nrow(u),myinfo$nwrkrs[[myinfo$id]])
    # multiply this thread's part of u with v, placing the product in
    # the
```
# corresponding part of w
w[myidxs,] <- u[myidxs,] %*% v[,] invisible(0)

# create test matrices
mgrmakevar(c2,"a",6,2)
mgrmakevar(c2,"b",2,6)
mgrmakevar(c2,"c",6,6)
# give them values
a[,] <- 1:12
b[,] <- 1 # all 1s
clusterExport(c2,"mmul") # send mmul() to the threads
clusterEvalQ(c2,mmul(a,b,c)) # run the threads
c[,] # check results

---

**barr**

*Barrier operation.*

**Description**

Standard barrier operation.

**Usage**

```r
barr()
```

**Details**

Standard barrier operation, to ensure that work done by one thread is ready before other threads make use of it. When a thread executes `barr()`, it will block until all threads have executed it.

**Author(s)**

Norm Matloff

---

**getidxs**

*Parallelizing work assignment.*

**Description**

Assigns to an `Rdsm` thread its portion of a set of indices, for the purpose of partitioning work to the threads.

**Usage**

```r
getidxs(m)
```
getmatrix

Arguments

m  The sequence 1:m will be partitioned, and one portion will be assigned to the calling thread.

Details

The range 1:m will be partitioned into r subranges, the i-th of which will be used by thread i to determine which work that thread has been assigned.

Value

The subrange assigned to the invoking thread.

Author(s)

Norm Matloff

getmatrix
Referencing a matrix via different forms.

Description

Returns a matrix, whether requested via an R matrix, bigmemory object, bigmemory descriptor or quoted name form.

Usage

getmatrix(m)

Arguments

m  Specification of the matrix, as either an R matrix, bigmemory object, bigmemory descriptor or quoted name form.

Details

This utility function enables writing general Rdsm code, specifying a matrix via different forms.

Value

The requested matrix.

Author(s)

Norm Matloff
Examples

```r
library(parallel)
c2 <- makeCluster(2)
mgrmakevar(c2, "u", 2, 2)  # u is a 2x2 bigmemory matrix
u[] <- 8  # fill u with 8s
u[]  # prints a 2x2 matrix of 8s
v <- getmatrix(u)  # get u and assign it to v
# u and v are both addresses, pointing to the same memory location
v[]  # prints all 8s
v[2,1] <- 3
v[]  # prints three 8s and a 3
u[]  # prints three 8s and a 3
w <- getmatrix("u")  # w will also be a copy of u
w[]  # same as u
```

loadex  
Sources example files

Description

Facilitates learning a new package, by sourcing any specified example file for an installed package.

Usage

```r
loadex(pkg, exfile=NULL, subdir="examples")
```

Arguments

- `pkg` Package name, quoted.
- `exfile` Desired example file name, if any, quoted.
- `subdir` Subdirectory name of the examples directory in the package.

Details

Loads the example file `exfile`, from the `subdir` directory within the tree where `pkg` is installed. If `exfile = NA`, the names of the files are returned, without any loading.

Value

See the case `exfile = NA` above.

Author(s)

Norm Matloff
makebarr

Create an Rdsm barrier.

Description

Creates an Rdsm barrier.

Usage

makebarr(cls, boost=F, barrback=F)

Arguments

| Argument    | Description
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cls</td>
<td>The snow cluster.</td>
</tr>
<tr>
<td>boost</td>
<td>Locks type. See mgrinit.</td>
</tr>
<tr>
<td>barrback</td>
<td>If TRUE, the count/sense variables related to the barrier will be placed in backing store.</td>
</tr>
</tbody>
</table>

Details

Run this from the manager (the R process from which you create the cluster) if you need a barrier. Only one barrier is allowed in an Rdsm program (but multiple calls are allowed). It is accessible from application code only through barr().

Author(s)

Norm Matloff

mgrinit

Initialize Rdsm

Description

Initializes Rdsm on the given parallel cluster.

Usage

mgrinit(cls, boost=F, barrback=F)

Arguments

| Argument    | Description
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cls</td>
<td>The parallel cluster.</td>
</tr>
<tr>
<td>boost</td>
<td>Lock functions. If TRUE, boostlock() and boostunlock() are used; otherwise backlock() and backunlock().</td>
</tr>
<tr>
<td>barrback</td>
<td>If TRUE, the count/sense variables associated with the barrier will be placed in backing store.</td>
</tr>
</tbody>
</table>
Details

Run this from the manager (the R process from which you create the cluster), before creating the shared variables with `mgrmakevar`. The initialization need be done only once for the life of the cluster.

If you put shared variables in backing store (`barrback = TRUE` in `mgrmakevar()`), or if you are on a Windows platform, you must have `boost = FALSE`.

Author(s)

Norm Matloff

---

**mgrmakelock**

*Create an Rdsm lock.*

Description

Creates an Rdsm lock.

Usage

`mgrmakelock(cls, lockname, boost=F)`

Arguments

- `cls`  The **parallel** cluster.
- `lockname`  Name of the lock, quoted.
- `boost`  If TRUE, boost locks will be used.

Details

Run this from the manager (the R process from which you create the cluster) if you need a lock. The lock is created, lockable/unlockable by all threads. If boost is TRUE, the variable will be of class `boost::mutex`; see the library **synchronicity** for details.

Author(s)

Norm Matloff
mgrmakevar

Create an Rdsm shared variable.

Description

Creates an Rdsm shared variable.

Usage

mgrmakevar(cls, varname, nr, nc, vartype="double", fs=FALSE, mgrcpy=TRUE, savedesc=TRUE)

Arguments

cls         The parallel cluster.
varname     Name of the shared variable, quoted. (The variable must be a matrix, though it could be 1x1 etc.)
nr           Number of rows in the variable.
nc           Number of columns in the variable.
vartype     Atomic R type of the variable, quoted, "double" by default.
fs           Place in backing store? FALSE by default.
mgrcpy      Place a copy of the shared variable on the manager node? TRUE by default.
savedesc    Save the bigmemory descriptor for this variable on disk.

Details

Run this from the manager (the R process from which you create the cluster). The shared variable will be created, readable/writable from all threads. The variable will be of class big.matrix; see the library bigmemory for details.

Author(s)

Norm Matloff

rdsmlock

Lock/unlock operations.

Description

Lock/unlock operations to avoid race conditions among the threads.

Usage

rdsmlock(lck)
rdsmunlock(lck)
Arguments

**Lock**

Lock name, quoted.

Details

Standard lock/unlock operations from the threaded coding world. When one thread executes `rdsmlock()`, any other thread attempting to do so will block until the first thread executes `rdsmunlock()`. If a thread does `rdsmlock()` on an unlocked lock, the thread call immediately returns and the thread continues.

These functions are set in the call to `mgrinit()` via the argument `boost` to either `boostlock` and `boosunlock()` or `backlock` and `backunlock()`, depending on whether you set `boost` to `TRUE` or `FALSE`, respectively.

Code should be written so that locks are used as sparingly as possible, since they detract from performance.

Author(s)

Norm Matloff

Examples

```r
## Not run:
# unreliable function
s <- function(n) {
  for (i in 1:n) {
    tot[1,1] <- tot[1,1] + 1
  }
}

library(parallel)
c2 <- makeCluster(2)
clusterExport(c2,"s")
mgrinit(c2)
mgrmakevar(c2,"tot",1,1)
tot[1,1] <- 0
clusterEvalQ(c2,s(1000))
tot[1,1] # should be 2000, but likely far from it

s1 <- function(n) {
  require(Rdsm)
  for (i in 1:n) {
    rdsmlock("totlock")
    tot[1,1] <- tot[1,1] + 1
    rdsmunlock("totlock")
  }
}

mgrmakevar(c2,"totlock")
tot[1,1] <- 0
```
### Description

Actions to propagate changes to file-backed Rdsm variables across a shared file system.

### Usage

- `readsync(varname)`
- `writesync(varname)`

### Arguments

- **varname**
  
  Name of the Rdsm variable, quoted.

### Details

This feature should be considered experimental, with poor performance and portability.

Suppose we have an Rdsm variable `x` which is in backing store (`fs = TRUE` in call to `mgrmakevar()`), and that we on a shared file system. (These functions are not needed if all threads are on the same machine.) When one Rdsm thread writes to `x`, the question here is when the updated value for `x` is visible to other Rdsm threads.

The answer may depend on the underlying file system. The functions `readsync()` and `writesync()` force the updates across the network. Normally one would call `readsync()` following `rdsmlock()` and call `writesync()` just before calling `rdsmunlock()`.

These should work on systems with "close-to-open cache coherency," as with the Network File System (NFS). On some systems, these functions should be unnecessary.

### Value

None.

### Author(s)

Norm Matloff
Examples

```r
library(parallel)
c2 <- makeCluster(2)
mgrinit(c2)
mgrmakevar(c2, "x", 1, 1, fs=TRUE)

clusterEvalQ(c2, me <- myinfo$id)
clusterEvalQ(c2, if (me==1) x[1,1] <- 3)
# force update on network
clusterEvalQ(c2, if (me==1) writesync("x"))
clusterEvalQ(c2, if (me==2) readsync("x"))
clusterEvalQ(c2, if (me==2) x[1,1] <- 8)
clusterEvalQ(c2, if (me==2) writesync("x"))
clusterEvalQ(c2, if (me==1) readsync("x"))
clusterEvalQ(c2, x[1,1]) # both should yield 8
```

stoprdsms  

Barrier operation.

Description

Standard barrier operation.

Usage

```r
stoprdsms(cls)
```

Arguments

```r
cls
```

Cluster from the `parallel` package.

Details

Shuts down the given parallel cluster, and removes any .desc files that had been created.

Author(s)

Norm Matloff
Index

*Topic parallel computation
   Rdsm-package, 2
*Topic shared memory
   Rdsm-package, 2
*Topic threads
   Rdsm-package, 2
*Topic
   Rdsm-package, 2
barr, 3, 4
getidxs, 3, 4
getmatrix, 3, 5
loadex, 6
makebarr, 7
mgrinit, 3, 7
mgrmakelock, 3, 8
mgrmakevar, 3, 9
Rdsm (Rdsm-package), 2
Rdsm-package, 2
rdsmlock, 3, 9
rdsmunlock, 3
rdsmunlock (rdsmlock), 9
readsync, 11
stoprdsm, 12
writesync (readsync), 11