A Quick Guide for the pbdDMAT Package

Drew Schmidt[1], Wei-Chen Chen[2], George Ostrouchov[1,2], Pragneshkumar Patel[1]

[1]Remote Data Analysis and Visualization Center
University of Tennessee,
Knoxville, TN, USA

[2]Computer Science and Mathematics Division,
Oak Ridge National Laboratory,
Oak Ridge, TN, USA

Contents

Acknowledgement

Ostrouchov, Schmidt, and Patel were supported in part by the project "NICS Remote Data Analysis and Visualization Center" funded by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center. Chen and Ostrouchov were supported in part by the project "Visual Data Exploration and Analysis of Ultra-large Climate Data" funded by U.S. DOE Office of Science under Contract No. DE-AC05-00OR22725.

This work used resources of National Institute for Computational Sciences at the University of Tennessee, Knoxville, which is supported by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center. This work also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This work used resources of the Newton HPC Program at the University of Tennessee, Knoxville.

We thank our colleague, Ed D'Azevedo from the Computational Mathematics Group, Computer Science and Mathematics Division, Oak Ridge National Laboratory (ORNL), for his discussions and illuminating advice using ScaLAPACK and distributed matrix computation.

We also thank Brian D. Ripley, Kurt Hornik, and Uwe Ligges from the R Core Team for discussing package release issues and helping us solve portability problems on different platforms.

Abstract

With the size of data ever growing, the use of multiple processors in a single analysis becomes more and more a necessity. The Programming Big Data (pbd) project attempts to address the R language's current shortcomings in parallel distributed computations. The **pbdDMAT** package for R provides high level S4 methods for the **pbdBASE** distributed matrix data type. These methods focus on computationally burdensome problems with block-cyclically distributed data, such as linear algebra, in such a way that the syntax for operating on new data types mimics R's syntax for ordinary matrices as closely as possible. This allows someone already familiar with R syntax to achieve vast speed improvements via parallelism with a mostly already familiar syntax, and without the need to learn complicated parallel programming techniques. Much of the heavy lifting is performed by the PBLAS and ScaLAPACK libraries. In addition to performance improvements through parallelism, use of this system with more than one processor allows the user to break R's local memory barrier, namely the requirement that a vector be indexed by a 32-bit integer, by only storing subsets of the vector on each processor.

1. Introduction

The **pbdDMAT** package (Programming with Big Data: Distributed Matrix Algebra Computation) (Schmidt *et al.* 2012b) is a (mostly) implicitly parallel system for doing distributed matrix computations in R. (R Core Team 2012) It offers numerous high level methods for a the **pbdBASE** (Schmidt *et al.* 2012a) distributed matrix type `ddmatrix` which intentionally, very closely resemble the existing R syntax for non-distributed matrices. Much of the heavy lifting — especially that involving distributed linear algebra — is handled by the well-known Fortran libraries Scalable Linear Algebra Package (ScaLAPACK) and the Parallel Basic Linear Algebra Subroutines (PBLAS). (Blackford *et al.* 1997)

Ordinarily, a user of these libraries would have to deal with a great many more headaches than the user of **pbdDMAT**. Of note, a user of the **pbdDMAT** system can achieve great speedups with only the most minimal interaction with the more cumbersome sides of ScaLAPACK, such as the MPI layer for ScaLAPACK, the Basic Linear Algebra Communication Subroutines (BLACS). (Dongarra and Whaley 1995) Local storage issues, descriptor vectors, and BLACS communications are very much still there, but almost all of these problems have been abstracted away for the user. In addition to offering copycat routines for linear algebra, the **pbdDMAT** package also offers many other routines that look native to R, but operate on these special distributed matrix data types.

The principal goal of the **pbdDMAT** package is to provide R users with access to extremely powerful distributed, implicitly parallel computation, all while preserving the friendly and familiar R syntax for these computations, so that effectively, much existing R code could used with this system with only trivial modifications, yet receive massive performance boosts.

1.1. Installation

The **pbdDMAT** package is available from the CRAN at `http://cran.r-project.org`, and can be installed via a simple

R Script

```
install.packages("pbdDMAT")
```

This assumes only that you have MPI installed and properly configured on your system. If the user can successfully install the package's three principal dependencies, **pbdMPI** (Chen *et al.* 2012a), **pbdSLAP** (Chen *et al.* 2012b), and **pbdBASE** (Schmidt *et al.* 2012a) (each available from the CRAN), then the installation for pbdDMAT should go smoothly. If you experience difficulty installing either these packages, you should see their documentation.

1.2. Package Examples

One can quickly get started with **pbdDMAT** by learning from the following five examples:

Shell Script

```
### Under command mode, run the demo with 2 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 2 Rscript -e "demo(a_reductions,
   package='pbdDMAT',ask=F,echo=F)"
mpiexec -np 2 Rscript -e "demo(b_matprod,
   package='pbdDMAT',ask=F,echo=F)"
mpiexec -np 2 Rscript -e "demo(c_solve,
   package='pbdDMAT',ask=F,echo=F)"
mpiexec -np 2 Rscript -e "demo(d_svd, package='pbdDMAT',ask=F,echo=F)"
mpiexec -np 2 Rscript -e "demo(e_cholesky,
   package='pbdDMAT',ask=F,echo=F)"
```

2. Class Methods

The **pbdDMAT** package contains many methods for the **pbdBASE** distributed matrix data type, each with a very similar (usually identical) look and feel to its R counterpart. Table 1 provides an incomplete list of the new methods for class `ddmatrix`. For details and other methods, see the official **pbdDMAT** documentation.

3. Using the pbdDMAT Package

Most of the difficulty in using **pbdDMAT** — creating and manipulating the distributed data type — actually lies in **pbdBASE**. Therefore we consider it crucial that the user of **pbdDMAT** read, at the least, the first few sections of the **pbdDMAT** vignette. Once the **pbdBASE** issues are handled, generally the user can just call `ddmatrix` methods exactly as he/she would with ordinary R matrices. So if the user wants to compute the singular value decomposition in parallel on a distributed matrix, the call is the same as if the user wants the singular value decomposition in serial of an ordinary R matrix. Some functions have extra headaches over their R counterparts, such as `apply()`, but every consideration has been taken to minimize the occurrence of these.

4. An Example

R S4 Method Overloading
+, −, ∗, /, ∧, %%, %/%
`t()`, `%*%`
`solve()`
`La.svd()`, `svd()`, `chol()`, `lu()`

(a) Linear Algebra Methods

R S4 Method Overloading
`scale()`
`cov()`
`prcomp()`

(b) Statistical Methods

R S4 Method Overloading
`min()`, `max()`
`sum()`, `prod()`
`mean()`, `median()`
`rowSums()`, `colSums()`
`rowMeans()`, `colMeans()`

(c) Reductions

R S4 Method Overloading
`apply()`
`round()`, `ceiling()`, `floor()`
`sqrt()`
`abs()`
`exp()`, `log()`

(d) Transformations

Table 1: Computational Methods for Class `ddmatrix`

Let's take a look at some example code. Here, we will do some distributed matrix multiplication, as well as solving some systems of equations. You probably should not use a large process grid for this problem. Anything bigger than 8×8 is *massive* overkill. A 2×2, 2×4, or 4×4 process grid (4, 8, or 16 processes respectively) should be more than plenty — so a humble laptop or desktop should more than suffice. You've got to crawl before you can drag race.

Throughout, we will preface distributed objects by a *d*, purely for pedagogical reasons. Non-distributed objects will not have any preface. So `x` is not distributed, but `dx` is.

To convince you that this new stuff is really doing the same things as the old stuff, we are going to randomly generate a 500×500 matrix on process 0, and then distribute that matrix across the process grid, using a 32×32 blocking dimension. If you are using more than 4 processes, you might consider backing that off to 16×16, but it's not really necessary; remember, the purpose here is to learn.

Generating Test Data

```
init.grid()

# Number of rows and columns to generate
nrows <- 5e2
ncols <- 5e2

mn <- 10
sdd <- 100

# ScaLAPACK blocking dimension
bldim <- c(4, 4)
```

```
# Generate data on process 0, then distribute to the others
if (comm.rank()==0) {
   x <- matrix(rnorm(n=nrows*ncols, mean=mn, sd=sdd),
      nrow=nrows, ncol=ncols)
   b <- matrix(rnorm(n=ncols*2, mean=mn, sd=sdd), nrow=ncols,
      ncol=2)
} else {
  x <- NULL
  b <- NULL
}

dx <- as.ddmatrix(x=x, bldim=bldim)
db <- as.ddmatrix(x=b, bldim=bldim)

# continued in the next block of code ...
```

All of this information is covered in the **pbdBASE** documentation and vignette (Schmidt *et al.* 2012c), and while all of the above code is in **pbdBASE**, it is good to go over this process again:

- **Load the package** with `library(pbdDMAT)`
- **Use `init.grid()` to initialize the process grid** and MPI communicator(s). You can optionally specify `nprow=` for the number of process rows here, and `npcol=` for the number of process columns. Not specifying means that the function will choose the "best" option for you, meaning a grid that is as close to square as possible for the number of processors you have given it.
- **Generate the** 500×500 **ordinary non-distributed R matrix** of random normal data with mean 10 and sd 100 on process 0. Here, we use the **pbdMPI** function `comm.rank()` to make sure that only process 0 generates the data. The other processes store `NULL` in `x`. Likewise, we do the same for the 500×2 vector `b`/`db`, the "right hand sides" for the systems we will be solving.
- **Distribute the data** from process 0 and store it as a distributed matrix named `dx`. Here we specify a blocking dimension of 32, so really 32×32. ScaLAPACK and PBLAS routines usually require square blocking, so while we could block in many other ways, like 32×16, this may not be a good plan. Whenever the `bldim=` option is present, specifying only a single integer `n` will always be equivalent to specifying `c(n, n)`.

Before continuing, it is good to reiterate that this is not an efficient way to do business if you are using many processes. You need to use multiple processes to either read in the matrix into pieces from disk in parallel, or you need to do random generation in parallel using multiple (perhaps all) processes.

Now that we have our data and have dealt with the **pbdBASE** side of things, it's pure easy street from here. Now we just forget that there is anything distributed at all going on and write our code exactly as we would with plain, vanilla R.

First, we will multiply the transpose of `dx` and calculate the inverse of this product, storing the result in `dx_inv`. Finally, we solve the system of equations with two right hand sides `dx_inv * solns = db`.

Simple Matrix Operations

```
# Computations in parallel
dx_inv <- solve( t(dx) %*% dx )
solns <- solve(dx_inv, db)

# continued in the next block of code ...
```

Notice that we're doing matrix computations just the same way you would with vanilla R. And to prove that it really is the same, we can undistribute our results and "check our work":

Comparing Results to R

```
# Undistribute solutions to process 0
pbd_dx_inv <- as.matrix(dx_inv, proc.dest=0)
pbd_solns <- as.matrix(solns, proc.dest=0)

# Compare our solution with R's --- not in parallel
if (comm.rank()==0) {
  r_x_inv <- solve( t(x) %*% x )
  r_solns <- solve(r_x_inv, b)

  print(all.equal(pbd_dx_inv, r_x_inv))
  print(all.equal(pbd_solns, r_solns))
}

# shut down the MPI communicators
finalize()
```

The above script is in the **pbdDMAT** directory, located at `inst/examples/dmat_vignette_eg.R`. To run the code, you would make a batch execution call. Say you have 4 processors you wish to use for this analysis. Then you could execute the script via the command:

```
# replace the 4 below with hoever many processors you actually want
    to use
mpirun -np 4 Rscript dmat_vignette_eg.R
```

from a terminal. If everything works correctly, then two `TRUE`'s will print to the terminal.

References

Blackford LS, Choi J, Cleary A, D'Azevedo E, Demmel J, Dhillon I, Dongarra J, Hammarling S, Henry G, Petitet A, Stanley K, Walker D, Whaley RC (1997). *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA. ISBN 0-89871-397-8 (paperback). URL http://netlib.org/scalapack/slug/scalapack_slug.html/.

Chen WC, Ostrouchov G, Schmidt D, Patel P, Yu H (2012a). "pbdMPI: Programming with Big Data – Interface to MPI." R Package, URL http://cran.r-project.org/package=pbdMPI.

Chen WC, Schmidt D, Ostrouchov G, Patel P (2012b). "pbdSLAP: Programming with Big Data – Scalable Linear Algebra Packages." R Package, URL http://cran.r-project.org/package=pbdSLAP.

Dongarra J, Whaley RC (1995). "A User's Guide to the BLACS." *Technical report*, University of Tennessee. UT-CS-95-281.

R Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.r-project.org/.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012a). "pbdBASE: Programming with Big Data – Core pbd Classes and Methods." R Package, URL http://cran.r-project.org/package=pbdBASE.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012b). "pbdDMAT: Programming with Big Data – Distributed Matrix Algebra Computation." R Package, URL http://cran.r-project.org/package=pbdDMAT.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012c). "A Quick Guide for the pbdBASE package." R Vignette, URL http://cran.r-project.org/package=pbdBASE.