Introducing R: from Your Laptop to HPC and Big Data

SC13

Drew Schmidt and George Ostrouchov

November, 2013

Affiliations and Support

The pbdR Core Team
http://r-pbd.org

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

Ostrouchov, Patel, and Schmidt 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.

[1] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN
[2] Remote Data Analysis and Visualization Center, University of Tennessee, Knoxville, TN

About This Presentation

Downloads

This presentation and supplemental materials are available at:

http://tutorial.r-pbd.org

About This Presentation

Conventions For Code Presentation

We will use two different forms of syntax highlighting. One for displaying results from an interactive R session:

```
R> "interactive"
[1] "interactive"
```

and one for presenting R scripts

```
"not interactive"
```

Contents

Contents

What is R?

- *lingua franca* for data analytics and statistical computing.
- Part programming language, part data analysis package.
- Dialect of S (Bell Labs).
- Syntax designed for data.

Who uses R?

Google, Pfizer, Merck, Bank of America, Shell[a], Oracle[b], Facebook, bing, Mozilla, okcupid[c], ebay[d], kickstarter[e], the New York Times[f]

---

[a] https://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=0

[b] http://www.oracle.com/us/corporate/features/features-oracle-r-enterprise-498732.html

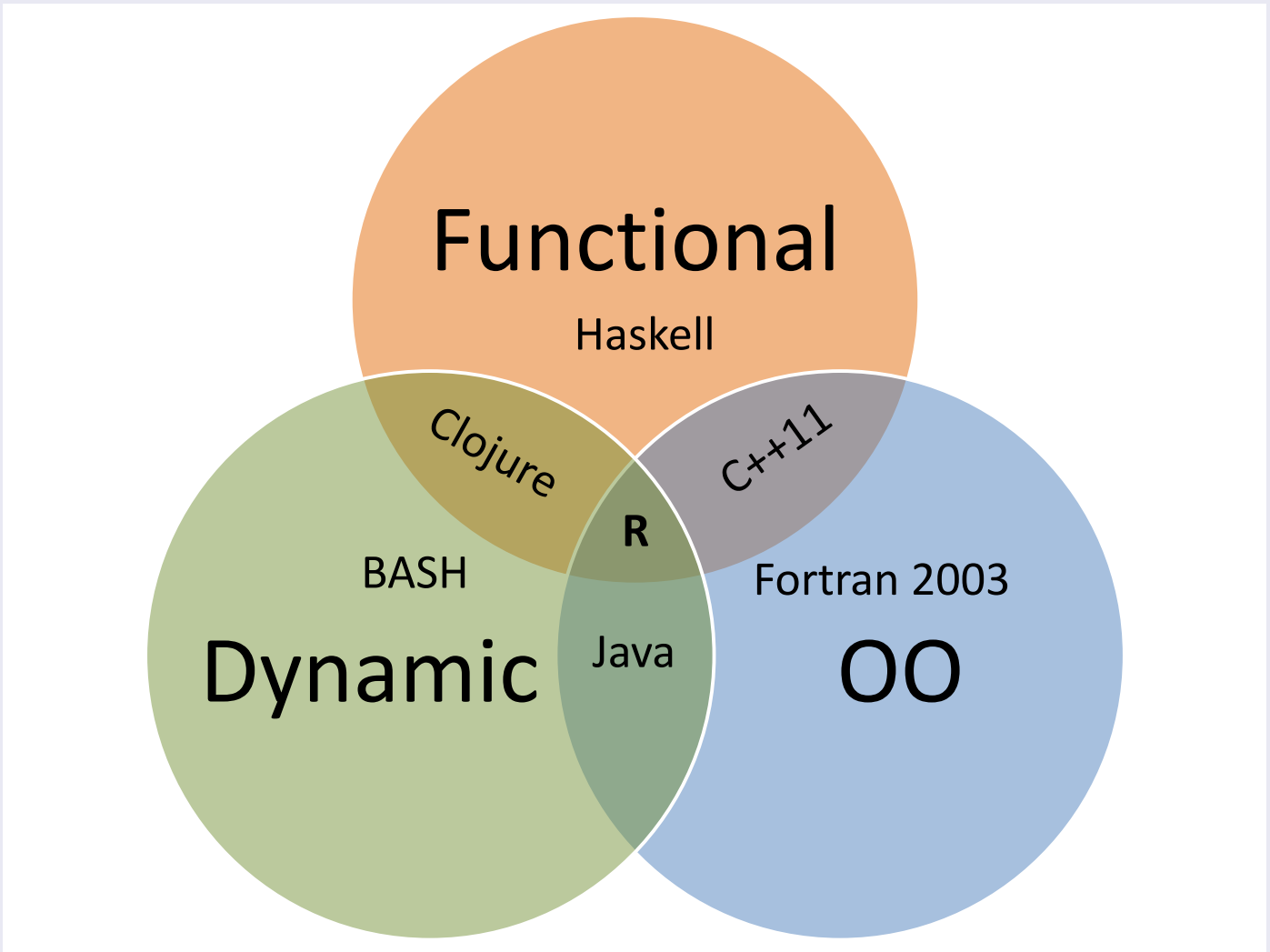
[c] http://www.revolutionanalytics.com/what-is-open-source-r/companies-using-r.php

[d] http://blog.revolutionanalytics.com/2012/09/using-r-in-production-industry-experts-share-their-experiences.html

[e] http://blog.revolutionanalytics.com/2012/09/kickstarter-facilitates-50m-in-indie-game-funding.html

[f] http://blog.revolutionanalytics.com/2012/05/nyt-charts-the-facebook-ipo-with-r.html

Language Paradigms



Data Types

- Storage: logical, int, double, double complex, character
- Structures: vector, matrix, array, list, dataframe
- Caveats: (Logical) `TRUE`, `FALSE`, `NA`

For the remainder of the tutorial, we will restrict ourselves to real number matrix computations.

Basics (1 of 2)

- The default method is to print:

```
1 R> sum
2 function (..., na.rm = FALSE)   .Primitive("sum")
```

- Use <- for assignment:

```
1 R> x <- 1
2 R> x+1
3 [1] 2
```

- Naming rules: mostly like C.
- R is case sensitive.
- We use . the way most languages use _, e.g., `La.svd()` instead of `La_svd()`.
- We use $ (sometimes @) the way most languages use .

Basics (2 of 2)

- Use ? or ?? to search help

```
1 R> ?set.seed
2 R> ?comm.set.seed
3 No documentation for 'comm.set.seed' in
    specified packages and libraries:
4 you could try '??comm.set.seed'
5 R> ??comm.set.seed
```

Addons and Extras

R has the Comprehensive R Archive Network (CRAN), which is a package repository like CTAN and CPAN.

From R

```
1 install.packages("pbdMPI") # install
2 library(pbdMPI)            # load
```

From Shell

```
1 R CMD INSTALL pbdMPI_0.1-6.tar.gz
```

Lists (1 of 1)

```
R> l <- list(a=1, b="a")
R> l
$a
[1] 1

$b
[1] "a"

R> l$a
[1] 1

R> list(x=list(a=1, b="a"), y=TRUE)
$x
$x$a
[1] 1

$x$b
[1] "a"


$y
[1] TRUE
```

Vectors and Matrices (1 of 2)

```
1  R> c(1, 2, 3, 4, 5, 6)
2  [1] 1 2 3 4 5 6
3
4  R> matrix(1:6, nrow=2, ncol=3)
5       [,1] [,2] [,3]
6  [1,]    1    3    5
7  [2,]    2    4    6
8
9  R> x <- matrix(1:6, nrow=2, ncol=3)
10
11 R> x[, -1]
12      [,1] [,2]
13 [1,]    3    5
14 [2,]    4    6
15
16 R> x[1, 1:2]
17 [1] 1 3
```

Vectors and Matrices (2 of 2)

```
R> dim(x)
[1] 2 3

R> dim(x) <- NULL
R> x
[1] 1 2 3 4 5 6

R> dim(x) <- c(3,2)
R> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Vector and Matrix Arithmetic (1 of 2)

```
R> 1:4 + 4:1
[1] 5 5 5 5

R> x <- matrix(0, nrow=2, ncol=3)

R> x + 1
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1

R> x + 1:3
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3
```

Vector and Matrix Arithmetic (2 of 2)

```
R> x <- matrix(1:6, nrow=2)

R> x*x
     [,1] [,2] [,3]
[1,]    1    9   25
[2,]    4   16   36

R> x %*% x
Error in x %*% x : non-conformable arguments

R> t(x) %*% x
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61

R> crossprod(x)
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61
```

Linear Algebra (1 of 2): Matrix Inverse

$$x_{n\times n} \text{ invertible} \iff \exists y_{n\times n}\,(xy = yx = Id_{n\times n})$$

```
1  R> x <- matrix(rnorm(5*5), nrow=5)
2  R> y <- solve(x)
3
4  R> round(x %*% y)
5        [,1] [,2] [,3] [,4] [,5]
6  [1,]    1    0    0    0    0
7  [2,]    0    1    0    0    0
8  [3,]    0    0    1    0    0
9  [4,]    0    0    0    1    0
10 [5,]    0    0    0    0    1
```

Linear Algebra (2 of 2): Singular Value Decomposition

$$x = U\Sigma V^T$$

```
R> x <- matrix(rnorm(2*3), nrow=3)
R> svd(x)
$d
[1] 2.4050716 0.3105008

$u
           [,1]       [,2]
[1,] 0.8582569 -0.1701879
[2,] 0.2885390  0.9402076
[3,] 0.4244295 -0.2950353

$v
            [,1]        [,2]
[1,] -0.05024326 -0.99873701
[2,] -0.99873701  0.05024326
```

More than just a Matlab clone...

- Data science (machine learning, statistics, data mining, ...) is mostly matrix algebra.

  So what about Matlab/Python/Julia/...?
- The one you prefer depends more on your "religion" rather than differences in capabilities.
- As a *data analysis* package, R is king.

Simple Statistics (1 of 2): Summary Statistics

```
R> x <- matrix(rnorm(30, mean=10, sd=3), nrow=10)

R> mean(x)
[1] 9.825177

R> median(x)
[1] 9.919243

R> sd(as.vector(x))
[1] 3.239388

R> colMeans(x)
[1]  9.661822 10.654686  9.159025

R> apply(x, MARGIN=2, FUN=sd)
[1] 2.101059 3.377347 4.087131
```

Simple Statistics (2 of 2): Sample Covariance

$$cov(x_{n\times p}) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(x_i - \mu_x)^T$$

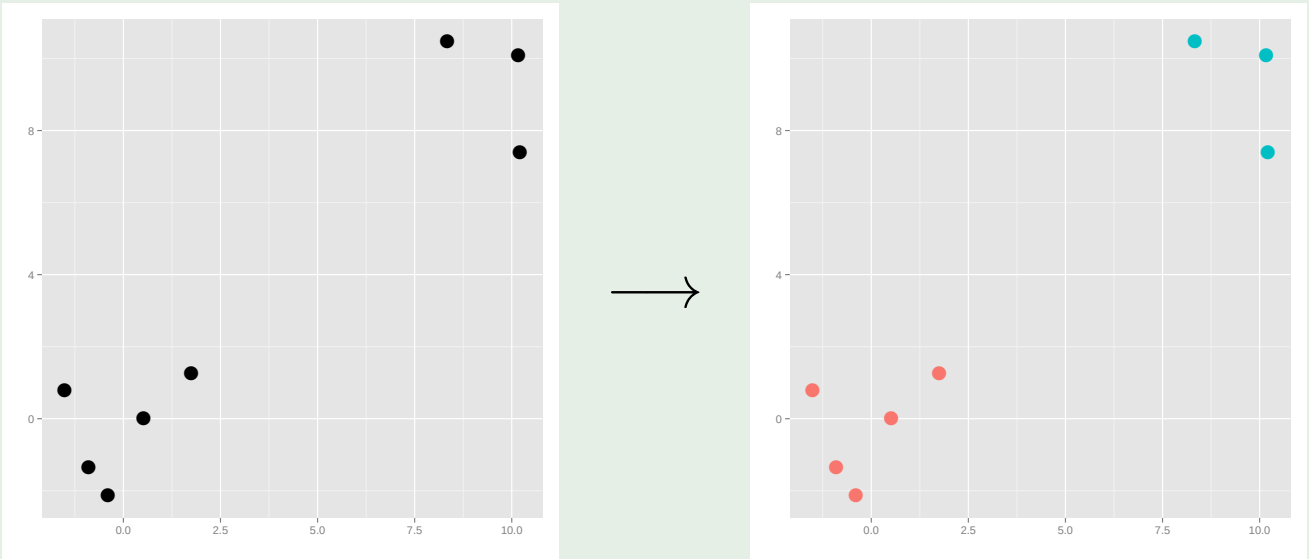
```
1  x <- matrix(rnorm(30), nrow=10)
2
3  # least recommended
4  cm <- colMeans(x)
5  crossprod(sweep(x, MARGIN=2, STATS=cm))
6
7  # less recommended
8  crossprod(scale(x, center=TRUE, scale=FALSE))
9
10 # recommended
11 cov(x)
```

Advanced Statistics (1 of 2): Principal Components

PCA = centering + scaling + rotation (via SVD)

```
1  R> x <- matrix(rnorm(30), nrow=10)
2
3  R> prcomp(x, retx=TRUE, scale=TRUE)
4  Standard deviations:
5  [1] 1.1203373 1.0617440 0.7858397
6
7  Rotation:
8                 PC1          PC2          PC3
9  [1,]  0.71697825 -0.3275365  0.6153552
10 [2,] -0.03382385  0.8653562  0.5000147
11 [3,]  0.69627447  0.3793133 -0.6093630
```

Advanced Statistics (2 of 2): k-Means Clustering




```
1 R> x <- rbind(matrix(rnorm(5*2, mean=0), ncol=2),
2               matrix(rnorm(3*2, mean=10), ncol=2))
```

Advanced Statistics (2 of 2): k-Means Clustering

```
R> kmeans(x, centers=2)
K-means clustering with 2 clusters of sizes 5, 3

Cluster means:
        [,1]       [,2]
1 -0.1080612 -0.2827576
2  9.5695365  9.3191892

Clustering vector:
[1] 1 1 1 1 1 2 2 2

Within cluster sum of squares by cluster:
[1] 14.675072  7.912641
 (between_SS / total_SS =  93.9 %)

Available components:

[1] "cluster"      "centers"      "totss"
    "withinss"     "tot.withinss"
[6] "betweenss"    "size"
```

Contents

The Iris Dataset

```
1 rm(list = ls())                    # Clean environment
2
3 head(iris)
4
5 ### Load data
6 X <- as.matrix(iris[, -5])         # Dimension 150 by 4
7 X.cid <- as.numeric(iris[, 5]) # True id
```

Standardizing

```
### Transformation and check
X.std <- scale(X)                    # Standardize
mu <- colMeans(X.std)                # Columns means are near 0
cov <- cov(X.std)                    # Diagonals are near 1
print(mu)
print(cov)
```

Projection Onto First 2 PC's

```
1  ### SVD
2  X.svd <- svd(X.std)
3
4  ### Project on column space of singular vectors
5  A <- X.std %*% diag(X.svd$d)
6  B <- X.std %*% X.svd$v
7  C <- prcomp(X.std)$x                    # A = B = C
8
9  X.prj <- C[, 1:2]                       # project onto first 2
       PC's
```

Clustering

```
### Clustering
set.seed(1234)                      # Set overall seed
X.kms <- kmeans(X.std, 3)           # K-means
X.kms
X.kms.cid <- X.kms$cluster          # Classification

library(EMCluster)                  # Model-based clustering
X.mbc <- init.EM(X.std, 3)          # Initial by em-EM
X.mbc
X.mbc.cid <- X.mbc$class            # Classification
```

Cluster Validation

```
### Validation
X.kms.adjR <- RRand(X.cid, X.kms.cid)$adjRand        # Adjusted Rand index
X.mbc.adjR <- RRand(X.cid, X.mbc.cid)$adjRand
```

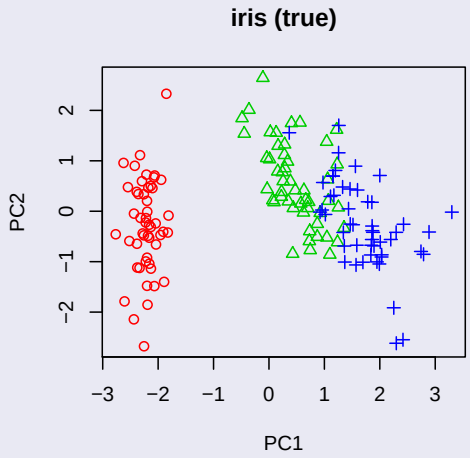
Cluster ID Variable

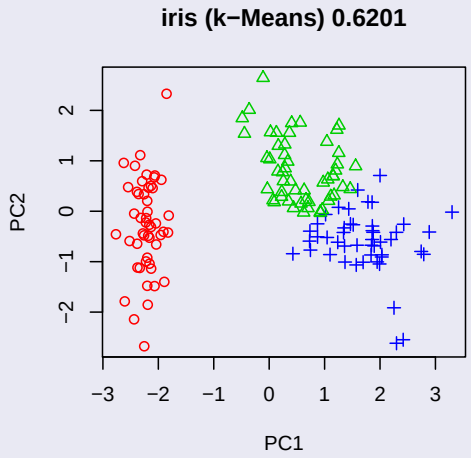
```
1 ### Swap classification id
2 X.kms.cid[X.kms.cid == 2] <- 4
3 X.kms.cid[X.kms.cid == 3] <- 2
4 X.kms.cid[X.kms.cid == 4] <- 3
```

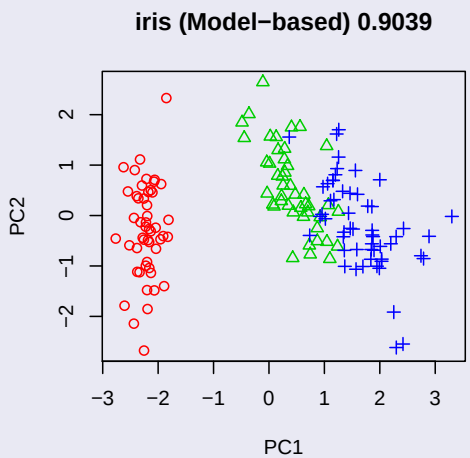
Plot

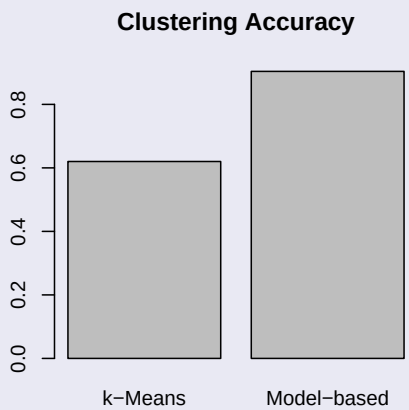
```
### Display on first 2 components
pdf("serial_plot.pdf")

par(mfrow = c(2, 2))
plot(X.prj, col = X.cid + 1, pch = X.cid,
     main = "iris (true)", xlab = "PC1", ylab = "PC2")
plot(X.prj, col = X.kms.cid + 1, pch = X.kms.cid,
     main = paste("iris (k-Means)", sprintf("%.4f",
         X.kms.adjR)),
     xlab = "PC1", ylab = "PC2")
plot(X.prj, col = X.mbc.cid + 1, pch = X.mbc.cid,
     main = paste("iris (Model-based)", sprintf("%.4f",
         X.mbc.adjR)),
     xlab = "PC1", ylab = "PC2")
accuracy <- c(X.kms.adjR, X.mbc.adjR)
names(accuracy) <- c("k-Means", "Model-based")
barplot(accuracy, main = "Clustering Accuracy")

dev.off()
```

Plot


iris (true)
PC2
PC1
−3 −2 −1 0 1 2 3
−2 −1 0 1 2


iris (k−Means) 0.6201
PC2
PC1
−3 −2 −1 0 1 2 3
−2 −1 0 1 2


iris (Model−based) 0.9039
PC2
PC1
−3 −2 −1 0 1 2 3
−2 −1 0 1 2


Clustering Accuracy
0.0 0.2 0.4 0.6 0.8
k−Means
Model−based

Contents

Problems with R

We *love* R! However...

- Slow.
- If you don't know what you're doing, it's *really* slow.
- Performance improvements usually for small machines.
- Very ram intensive.
- Chokes on big data.

Problems with R: Big Data

One of R's biggest problems is an indexing limitation:

- Any one R object must (at present) be indexed by a 32-bit integer.
- Largest vector/matrix: 16gb
- Largest square matrix: 46340×46340

R and Parallelism

The solution to many of R's problems is parallelism. However . . .

What we have

1. Mostly serial.
2. Parallelism mostly not distributed.
3. Data parallelism mostly explicit.

What we want

1. Mostly parallel.
2. Mostly distributed.
3. Mostly implicit.

Programming with Big Data in R (pbdR)

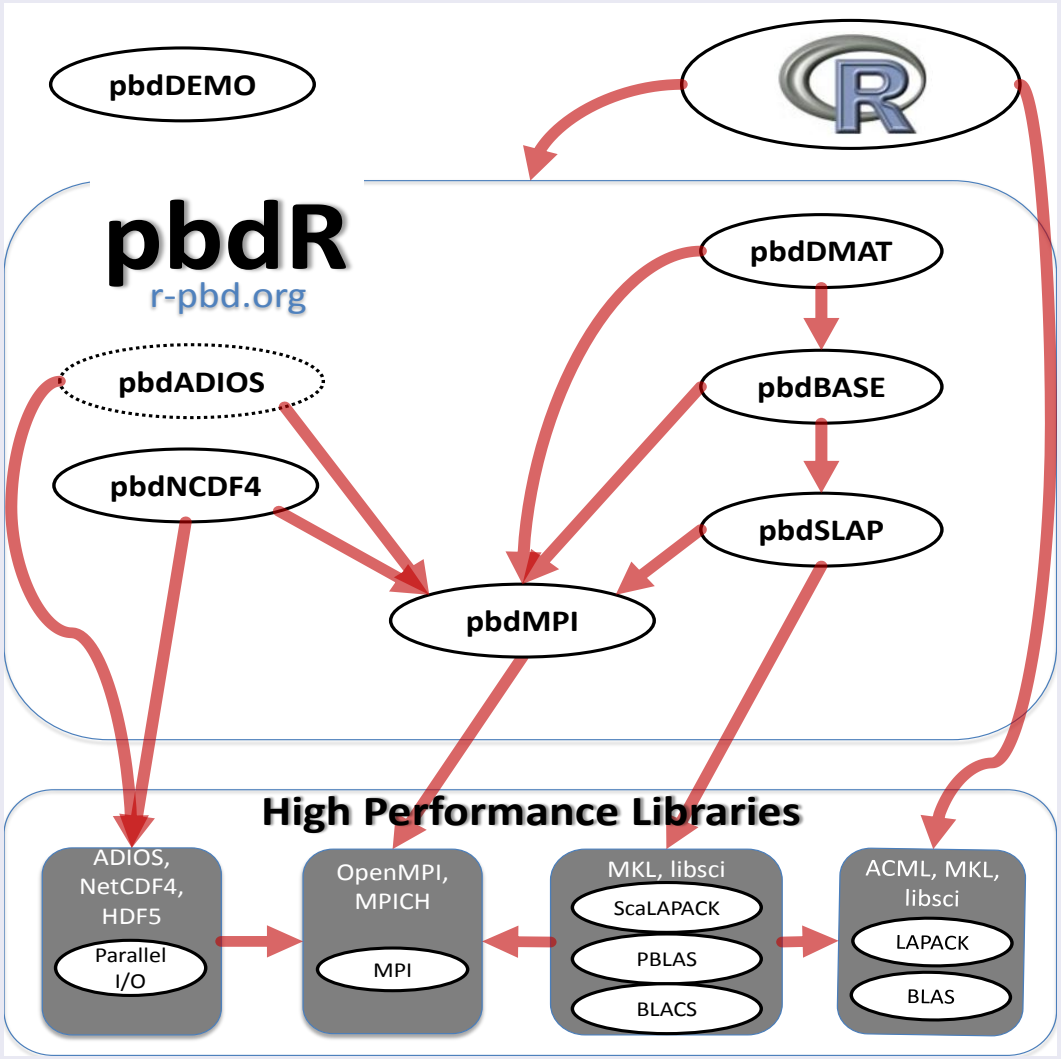
Goals: *Productivity, Portability, Performance*

Our Approach:

- Series of *free*[a] R packages.
- Scalable, big data analytics with high-level syntax.
- Implicit management of distributed data details.
- Methods have syntax *identical* to R.
- Powered by state of the art numerical libraries (MPI, ScaLAPACK, PBLAS, BLACS, LAPACK, BLAS, . . .)

[a] MPL, BSD, and GPL licensed

pbdR Packages



pbdR Packages — http://code.r-pbd.org

Released to CRAN:

- **pbdMPI**: MPI bindings (explicit, low-level)
- **pbdSLAP**: Foreign library (just install it, nothing to use)
- **pbdBASE**: Compiled code (used by DMAT, also for devs)
- **pbdDMAT**: Distributed matrices (mostly implicit, high-level)
- **pbdNCDF4**: Parallel NetCDF4 reader
- **pbdDEMO**: Package demonstrations, examples, vignette written in textbook style

Future Development:

- **pbdADIOS**: Wrappers for ADIOS middleware
- Profiling tools
- Client/server interface for interactive sessions
- Something for you. . . ?

OAK RIDGE National Laboratory

SPMD

The pbdR Packages enable high-level "Single Program/Multiple Data" (SPMD) programming:

- SPMD is a programming *paradigm*.
- Arguably the simplest extension of serial programming.
- Sort of like trying to explain breathing . . .
- Not to be confused with SIMD.
- SPMD utilizes MIMD architecture computers.
- Only one program is written, executed in batch independently on all processors.
- Different processors are autonomous; there is no manager.

SPMD

SPMD codes are run in batch (non-interactively):

From the Shell

```
1 mpirun -np 4 Rscript my_script.R
```

Example Syntax

```
1 x <- x[-1, 2:5]
2 xtx <- t(x) %*% x
3 ans <- svd(solve(xtx))
```

Look familiar?

The above runs on 1 core with R or 10,000 cores with pbdR

Installation

Installing pbdR is about as easy as possible, and generally amounts to:

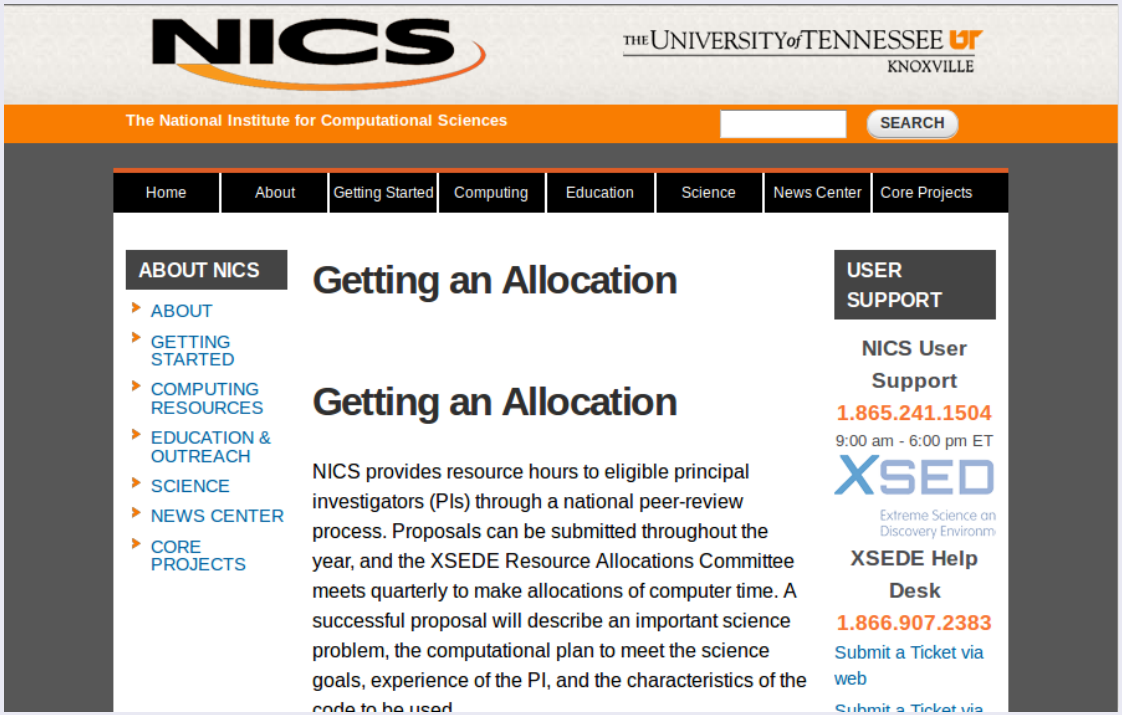
```
1 install.packages(pbdMPI)
2 install.packages(pbdNCDF4)
3 install.packages(pbdSLAP)
4 install.packages(pbdBASE)
5 install.packages(pbdDMAT)
6 install.packages(pbdDEMO)
```

But this assumes you have MPI installed on your system...

NICS Allocation

Instead, consider getting an allocation on Nautilus:

http://www.nics.tennessee.edu/getting-an-allocation



Brief Intermission

Brief Intermission

Questions? Comments?

Don't forget to talk to us at our discussion group:
http://group.r-pbd.org/

Don't have an allocation with us?
http://www.nics.tennessee.edu/getting-an-allocation

Contents

Message Passing Interface (MPI)

- *MPI*: Standard for managing communications (data and instructions) between different nodes/computers.
- *Implementations*: OpenMPI, MPICH2, Cray MPT, ...
- Enables parallelism on distributed machines.
- *Communicator*: manages communications between processors.

Common MPI Operations (1 of 2)

- **Managing a Communicator**: Create and destroy communicators.
  `init()` — initialize communicator
  `finalize()` — shut down communicator(s)

- **Rank query**: determine the processor's position in the communicator.
  `comm.rank()` — "who am I?"
  `comm.size()` — "how many of us are there?"

- **Barrier**: "computation wall"; no processor can proceed until *all* processors can proceed.
  `barrier()`

Quick Example 1

Rank Query

```
1 library(pbdMPI, quiet = TRUE)
2 init()
3
4 myRank <- comm.rank()
5 comm.print(myRank, all.rank=TRUE)
6
7 finalize()
```

Sample Output

```
1 COMM.RANK = 0
2 [1] 0
3 COMM.RANK = 1
4 [1] 1
```

Common MPI Operations (2 of 2)

- **Reduction**: each processor has a number `x.spmd`; add all of them up, find the largest/smallest, ...
  `reduce(x.spmd, op='sum')` — reduce to one
  `allreduce(x.spmd, op='sum')` — reduce to all

- **Gather**: each processor has a number; create a new object on some processor containing all of those numbers.
  `gather(x.spmd)` — gather to one
  `allgather(x.spmd)` — gather to all

- **Broadcast**: one processor has a number `x.spmd` that every other processor should also have.
  `bcast(x.spmd)`

Quick Example 2

```
1  library(pbdMPI, quiet = TRUE)
2  init()
3
4  comm.set.seed(diff=TRUE)
5
6  n <- sample(1:10, size=1)
7
8  sm <- allreduce(n, op='sum')
9  comm.print(sm)
10
11 gt <- allgather(n)
12 comm.print(unlist(gt))
13
14 finalize()
```

Sample Output

```
1  COMM.RANK = 0
2  [1] 10
3  COMM.RANK = 0
4  [1] 2 8
```

The SPMD Data Structure

Throughout the examples, we will make use of the SPMD distributed matrix structure.

1. SPMD is *distributed*. No one processor owns all of the matrix.
2. SPMD is *non-overlapping*. Any row owned by one processor is owned by no other processors.
3. SPMD is *row-contiguous*. If a processor owns one element of a row, it owns the entire row.
4. SPMD is globally *row-major*, locally *column-major*.
5. The last row of the local storage of a processor is adjacent (by global row) to the first row of the local storage of next processor (by communicator number) that owns data.
6. SPMD is (relatively) easy to understand, but can lead to bottlenecks if you have many more columns than rows.

Understanding SPMD: Global Matrix

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\ x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\ x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\ x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{bmatrix}_{9 \times 9}$$

Processors = 0 1 2 3 4 5

Understanding SPMD: Load Balanced SPMD

$$x = \left[\begin{array}{ccccccccc}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\
\hline
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\
\hline
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\
\hline
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\
\hline
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\
\hline
x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99}
\end{array}\right]_{9\times 9}$$

Processors = 0 1 2 3 4 5

Understanding SPMD: Local View

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \end{bmatrix}_{2\times 9}$$

$$\begin{bmatrix} x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \end{bmatrix}_{2\times 9}$$

$$\begin{bmatrix} x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \end{bmatrix}_{2\times 9}$$

$$\begin{bmatrix} x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \end{bmatrix}_{1\times 9}$$

$$\begin{bmatrix} x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \end{bmatrix}_{1\times 9}$$

$$\begin{bmatrix} x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{bmatrix}_{1\times 9}$$

Processors = 0 1 2 3 4 5

Understanding SPMD: Non-Balanced SPMD

$$x = \left[\begin{array}{ccccccccc}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\
\hline
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\
\hline
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\
\hline
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\
x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99}
\end{array}\right]_{9\times 9}$$

Processors = 0 1 2 3 4 5

Understanding SPMD: Local View

$$
\begin{bmatrix} \end{bmatrix}_{0\times 9}
\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \end{bmatrix}_{4\times 9}
\begin{bmatrix} x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \end{bmatrix}_{2\times 9}
\begin{bmatrix} x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \end{bmatrix}_{1\times 9}
\begin{bmatrix} \end{bmatrix}_{2\times 9}
\begin{bmatrix} x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\ x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{bmatrix}_{2\times 9}
$$

Processors = 0 1 2 3 4 5

Quick Comments for Using pbdMPI

1. Start by loading the package:

```
1 library(pbdMPI, quiet = TRUE)
```

2. Always initialize before starting and finalize when finished:

```
1 init()
2 # ...
3 finalize()
```

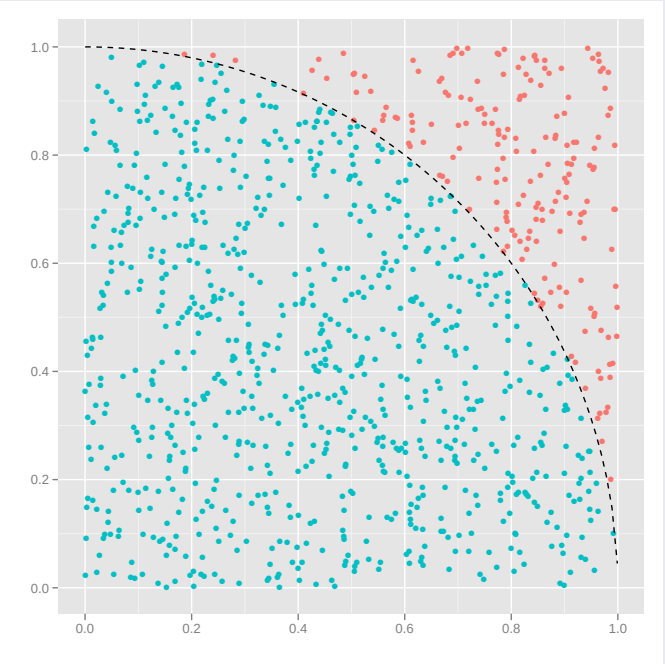
3. Use `comm.set.seed(diff=TRUE)` to generate independent streams by L'Ecuyer's method. Use `comm.set.seed(diff=FALSE)` to set a common seed among all processors.
4. Local pieces of SPMD distributed objects will be given the suffix `.spmd` to visually help distinguish them from global objects. This suffix carries no semantic meaning.

Contents

Example 1: Monte Carlo Simulation

Sample N uniform observations (x_i, y_i) in the unit square $[0,1] \times [0,1]$. Then

$$\pi \approx 4\left(\frac{\#\ Inside\ Circle}{\#\ Total}\right) = 4\left(\frac{\#\ \text{Blue}}{\#\ \text{Blue} + \#\ \text{Red}}\right)$$



Example 1: Monte Carlo Simulation SPMD Algorithm

1. Let n be big-ish; we'll take $n = 50,000$.
2. Generate an $n \times 2$ matrix x of standard uniform observations.
3. Count the number of rows satisfying $x^2 + y^2 \leq 1$
4. Ask everyone else what their answer is; sum it all up.
5. Take this new answer, multiply by 4 and divide by n
6. If my rank is 0, print the result.

Example 1: Monte Carlo Simulation Code

Serial Code

```
1  N <- 50000
2  X <- matrix(runif(N * 2), ncol=2)
3  r <- sum(rowSums(X^2) <= 1)
4  PI <- 4*r/N
5  print(PI)
```

Parallel Code

```
1  library(pbdMPI, quiet = TRUE)
2  init()
3  comm.set.seed(diff=TRUE)
4
5  N.spmd <- 50000 / comm.size()
6  X.spmd <- matrix(runif(N.spmd * 2), ncol = 2)
7  r.spmd <- sum(rowSums(X.spmd^2) <= 1)
8  r <- allreduce(r.spmd)
9  PI <- 4*r/(N.spmd * comm.size())
10 comm.print(PI)
11
12 finalize()
```

Note

For the remainder, we will exclude loading, init, and finalize calls.

Example 2: Sample Covariance

$$cov(x_{n\times p}) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(x_i - \mu_x)^T$$

Example 2: Sample Covariance SPMD Algorithm

1. Determine the total number of rows N.
2. Compute the vector of column means of the full matrix.
3. Subtract each column's mean from that column's entries in each local matrix.
4. Compute the crossproduct locally and reduce.
5. Divide by $N - 1$.

Example 2: Sample Covariance Code

Serial Code

```
1 N <- nrow(X)
2 mu <- colSums(X) / N
3
4 X <- sweep(X, STATS=mu, MARGIN=2)
5 Cov.X <- crossprod(X.spmd) / (N-1)
6
7 print(Cov.X)
```

Parallel Code

```
1 N <- allreduce(nrow(X.spmd), op="sum")
2 mu <- allreduce(colSums(X.spmd) / N, op="sum")
3
4 X.spmd <- sweep(X.spmd, STATS=mu, MARGIN=2)
5 Cov.X <- allreduce(crossprod(X.spmd), op="sum") / (N-1)
6
7 comm.print(Cov.X)
```

Example 3: Linear Regression

Find β such that

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

When $\mathbf{X}$ is full rank,

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Example 3: Linear Regression SPMD Algorithm

1. Locally, compute $tx = x^T$
2. Locally, compute $A = tx * x$. Query every other processor for this result and sum up all the results.
3. Locally, compute $B = tx * y$. Query every other processor for this result and sum up all the results.
4. Locally, compute $A^{-1} * B$

Example 3: Linear Regression Code

Serial Code

```
1 tX <- t(X)
2 A <- tX %*% X
3 B <- tX %*% y
4
5 ols <- solve(A) %*% B
```

Parallel Code

```
1 tX.spmd <- t(X.spmd)
2 A <- allreduce(tX.spmd %*% X.spmd, op = "sum")
3 B <- allreduce(tX.spmd %*% y.spmd, op = "sum")
4
5 ols <- solve(A) %*% B
```

Contents

Distributed Matrices

Most problems in data science

- Data structure: block-cyclic matrix distributed across a 2-dimensional grid of processors.
- No single processor should hold all of the data.
- Very robust, but very confusing data structure.

Distributed Matrices

The data structure is a special R class (in the OOP sense) called `ddmatrix`. It is the "under the rug" storage for a block-cyclic matrix distributed onto a 2-dimensional processor grid.

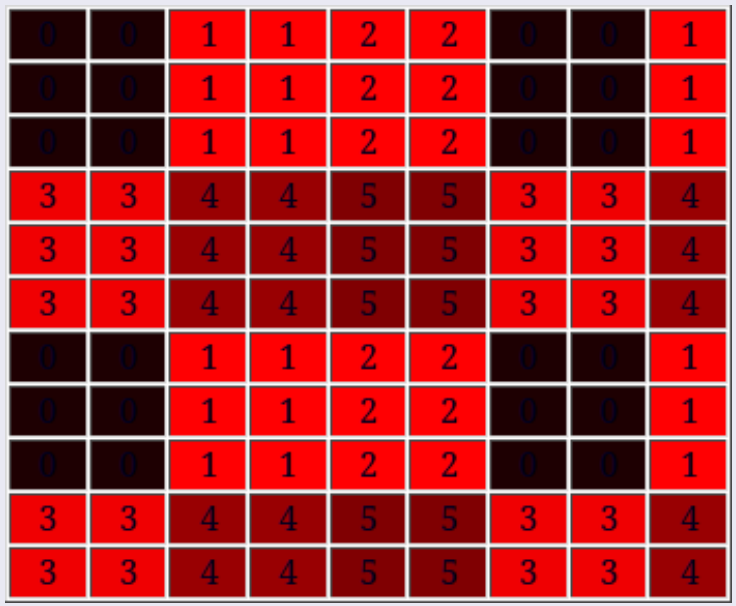
$$\texttt{ddmatrix} = \begin{cases} \textbf{Data} & \text{S4 local submatrix, an R matrix} \\ \textbf{dim} & \text{S4 dimension of the global matrix, a numeric pair} \\ \textbf{ldim} & \text{S4 dimension of the local submatrix, a numeric pair} \\ \textbf{bldim} & \text{S4 ScaLAPACK blocking factor, a numeric pair} \\ \textbf{CTXT} & \text{S4 BLACS context, an numeric singleton} \end{cases}$$

with prototype

$$\texttt{new("ddmatrix")} = \begin{cases} \textbf{Data} & = \texttt{matrix(0.0)} \\ \textbf{dim} & = \texttt{c(1,1)} \\ \textbf{ldim} & = \texttt{c(1,1)} \\ \textbf{bldim} & = \texttt{c(1,1)} \\ \textbf{CTXT} & = \texttt{0.0} \end{cases}$$

Distributed Matrices: The Data Structure

Example: an 11×9 matrix is distributed with a "block-cycling" factor of 3×2 on a 2×3 processor grid:



$$= \begin{cases} \textbf{Data} & = \texttt{matrix(...)} \\ \textbf{dim} & = \texttt{c(11, 9)} \\ \textbf{ldim} & = \texttt{c(...)} \\ \textbf{bldim} & = \texttt{c(3, 2)} \\ \textbf{CTXT} & = 0 \end{cases}$$

See http://acts.nersc.gov/scalapack/hands-on/datadist.html

Understanding Dmat: Global Matrix

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\ x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\ x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\ x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{bmatrix}_{9\times 9}$$

$$\text{Processor grid} = \left| \begin{matrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{matrix} \right| = \left| \begin{matrix} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \end{matrix} \right|$$

Understanding DMAT: Distributed with `bldim` = (2,2)

$$x = \left[\begin{array}{cc|cc|cc|cc|c} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\ \hline x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\ \hline x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\ x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\ \hline x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{array}\right]_{9\times 9}$$

$$\text{Processor grid} = \left|\begin{array}{ccc} 0 & 1 & 2 \\ 3 & 4 & 5 \end{array}\right| = \left|\begin{array}{ccc} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \end{array}\right|$$

Understanding DMAT: Local View

$$\left[\begin{array}{cc|cc} x_{11} & x_{12} & x_{17} & x_{18} \\ x_{21} & x_{22} & x_{27} & x_{28} \\ \hline x_{51} & x_{52} & x_{57} & x_{58} \\ x_{61} & x_{62} & x_{67} & x_{68} \\ \hline x_{91} & x_{92} & x_{97} & x_{98} \end{array}\right]_{5\times4} \left[\begin{array}{cc|c} x_{13} & x_{14} & x_{19} \\ x_{23} & x_{24} & x_{29} \\ \hline x_{53} & x_{54} & x_{59} \\ x_{63} & x_{64} & x_{69} \\ \hline x_{93} & x_{94} & x_{99} \end{array}\right]_{5\times3} \left[\begin{array}{cc} x_{15} & x_{16} \\ x_{25} & x_{26} \\ \hline x_{55} & x_{56} \\ x_{65} & x_{66} \\ \hline x_{95} & x_{96} \end{array}\right]_{5\times2}$$

$$\left[\begin{array}{cc|cc} x_{31} & x_{32} & x_{37} & x_{38} \\ x_{41} & x_{42} & x_{47} & x_{48} \\ \hline x_{71} & x_{72} & x_{77} & x_{78} \\ x_{81} & x_{82} & x_{87} & x_{88} \end{array}\right]_{4\times4} \left[\begin{array}{cc|c} x_{33} & x_{34} & x_{39} \\ x_{43} & x_{44} & x_{49} \\ \hline x_{73} & x_{74} & x_{79} \\ x_{83} & x_{84} & x_{89} \end{array}\right]_{4\times3} \left[\begin{array}{cc} x_{35} & x_{36} \\ x_{45} & x_{46} \\ \hline x_{75} & x_{76} \\ x_{85} & x_{86} \end{array}\right]_{4\times2}$$

$$\text{Processor grid} = \left|\begin{array}{ccc} 0 & 1 & 2 \\ 3 & 4 & 5 \end{array}\right| = \left|\begin{array}{ccc} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \end{array}\right|$$

The DMAT Data Structure

1. DMAT is *distributed*. No one processor owns all of the matrix.
2. DMAT is *non-overlapping*. Any piece owned by one processor is owned by no other processors.

---

3. DMAT can be row-contiguous or not, depending on the blocking factor used.
4. DMAT is locally column-major and globally, it depends...
5. If `bldim[2] > ncol(X)` and `bldim[1] > nrow(X) / comm.size()` then SPMD is a generalization of DMAT. Otherwise, no relation.
6. DMAT is confusing, but very robust.

Pros and Cons of This Data Structure

Pros

- Fast for distributed matrix computations

Cons

- Literally everything else

This is why we hide most of the distributed details.

The details are there if you want them (you don't want them).

Distributed Matrix Methods

pbdDMAT has over 100 methods with *identical* syntax to R:

- `` `[` ``, `rbind()`, `cbind()`, ...
- `lm.fit()`, `prcomp()`, `cov()`, ...
- `` `%*%` ``, `solve()`, `svd()`, `norm()`, ...
- `median()`, `mean()`, `rowSums()`, ...

Serial Code

```
1 cov(x)
```

Parallel Code

```
1 cov(x)
```

Comparing pbdMPI and pbdDMAT

- **pbdMPI** is MPI + some sugar.
- The `SPMD` data structure is not the only thing **pbdMPI** can handle (just a useful convention).
- **pbdDMAT** is more of a software package.
- The block-cyclic `DMAT` structure *must* be used for **pbdDMAT**.

Quick Comments for Using pbdDMAT

1. Start by loading the package:

```
1 library(pbdDMAT, quiet = TRUE)
```

2. Always initialize before starting and finalize when finished:

```
1 init.grid() # auto-calls pbdMPI's init()
2 # ...
3 finalize()
```

3. Once you have your data in the right format, the code becomes identical to serial R code. The hard part is getting the data in the right format...
4. Distributed `DMAT` objects will be given the suffix `.dmat` to visually help distinguish them from global objects. This suffix carries no semantic meaning.

Sample Covariance

Serial Code

```
1 Cov.X <- cov(X)
2 print(Cov.X)
```

Parallel Code

```
1 Cov.X <- cov(X)
2 print(Cov.X)
```

Linear Regression Code

Serial Code

```
1 tX <- t(X)
2 A <- tX %*% X
3 B <- tX %*% y
4
5 ols <- solve(A) %*% B
6
7 # or
8 ols <- lm.fit(X, y)
```

Parallel Code

```
1 tX <- t(X)
2 A <- tX %*% X
3 B <- tX %*% y
4
5 ols <- solve(A) %*% B
6
7 # or
8 ols <- lm.fit(X, y)
```

Generating Random Data

Using randomly generated matrices is the best way to "get your feet wet" with the pbd tools. You can do this in 2 ways:

1. Generate a global matrix and distribute it.
2. Generate locally only what is needed.

Example 1: Random Distributed Matrix Generation

Generate a global matrix and distribute it

```
1  library(pbdDMAT, quiet=TRUE)
2  init.grid()
3
4  # Common global on all processors --> distributed
5  comm.set.seed(diff=FALSE)
6  x <- matrix(rnorm(100), nrow=10, ncol=10)
7  x.dmat <- as.ddmatrix(x)
8
9  # Global on processor 0 --> distributed
10 if (comm.rank()==0){
11   x <- matrix(rnorm(100), nrow=10, ncol=10)
12 } else {
13   x <- NULL
14 }
15 x.dmat <- as.ddmatrix(x)
16
17 finalize()
```

Example 2: Random Distributed Matrix Generation

Generate locally only what is needed

```
1 library(pbdDMAT, quiet=TRUE)
2 init.grid()
3
4 comm.set.seed(diff = TRUE) # good seeds via rlecuyer
5 x.dmat <- ddmatrix("rnorm", nrow=10, ncol=10)
6
7 finalize()
```

Example 3: Random Distributed Matrix Generation

Generate locally only what is needed

```
1 library(pbdDMAT, quiet=TRUE)
2 init.grid()
3
4 zero.dmat <- ddmatrix(0, nrow=100, ncol=100)
5 id.dmat <- diag(1, nrow=100, ncol=100)
6
7 finalize()
```

Example 4: Random Distributed Matrix Generation

Convert between SPMD and DMAT

```
1  library(pbdDEMO, quiet=TRUE)
2  init.grid()
3
4  comm.set.seed(diff = TRUE)
5
6  N.spmd <- 1 + comm.rank()
7  X.spmd <- matrix(rnorm(N.spmd * 3), ncol = 3)
8
9  # convert SPMD to DMAT
10 X.dmat <- spmd2dmat(X.spmd)
11
12 # convert DMAT to SPMD
13 new.X.spmd <- dmat2spmd(X.dmat)
14
15 # undistribute
16 X <- as.matrix(X.dmat)
17
18 finalize()
```

Distributed Matrices

pbdDEMO contains many other examples of reading and managing SPMD and DMAT data

Contents

The Iris Dataset

```
1  rm(list = ls())                          # Clean environment
2
3  library(pbdDMAT, quiet = TRUE)           # Load library
4  init.grid()
5  if(comm.size() != 4)
6    comm.stop("4 processors are required.")
7
8  ### Load data
9  X <- as.matrix(iris[, -5])               # Dimension 150 by 4
10 X.cid <- as.numeric(iris[, 5])           # True id
11
12 ### Convert to ddmatrix
13 X.dmat <- as.ddmatrix(X)
```

Standardizing

```
1 ### Standardized
2 X.std <- scale(X.dmat)
3 mu <- as.matrix(colMeans(X.std))
4 cov <- as.matrix(cov(X.std))
5 comm.print(mu)
6 comm.print(cov)
```

Projection Onto First 2 PC's

```
1 ### SVD
2 X.svd <- svd(X.std)
3
4 ### Project on column space of singular vectors
5 A <- X.svd$u %*% diag(X.svd$d, type="ddmatrix")
6 B <- X.std %*% X.svd$v                  # A ~ B
7 X.prj <- as.matrix(A[, 1:2])           # Only useful for plot
```

Clustering

```
### Clustering
library(pmclust, quiet = TRUE)
comm.set.seed(123, diff = TRUE)

X.dmat <- X.std
PARAM.org <- set.global.dmat(K = 3)        # Preset storage
.pmclustEnv$CONTROL$debug <- 0             # Disable debug
    messages
PARAM.org <- initial.center.dmat(PARAM.org)
PARAM.kms <- kmeans.step.dmat(PARAM.org) # K-means
X.kms.cid <- as.vector(.pmclustEnv$CLASS.dmat)
```

Cluster Validation

```
1 ### Validation
2 X.kms.adjR <- EMCluster::RRand(X.cid, X.kms.cid)$adjRand
3 comm.print(X.kms.adjR)
```

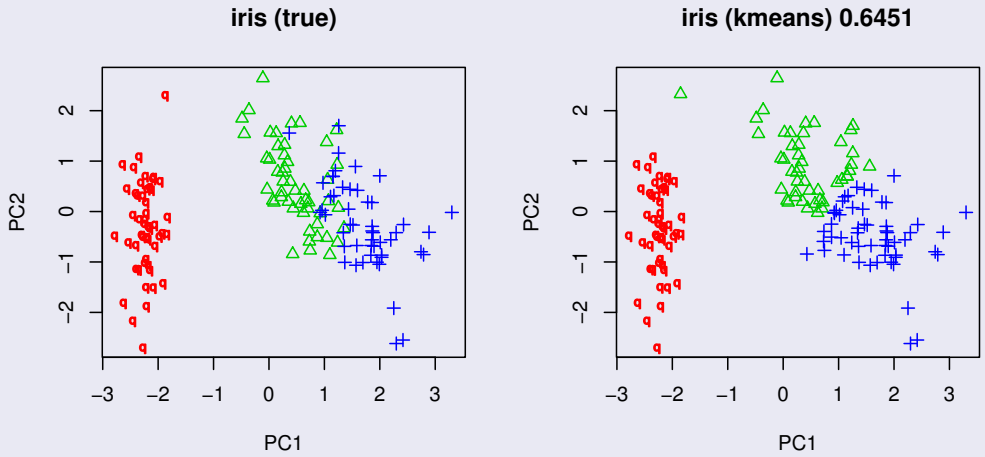
Cluster ID Variable

```
1 ### Swap classification id
2 tmp <- X.kms.cid
3 X.kms.cid[tmp == 1] <- 3
4 X.kms.cid[tmp == 2] <- 1
5 X.kms.cid[tmp == 3] <- 2
```

Plot

```
1  ### Display on first 2 components
2  if(comm.rank() == 0){
3    pdf("dmat_plot.pdf")
4
5    par(mfrow = c(2, 2))
6    plot(X.prj, col = X.cid + 1, pch = X.cid,
7         main = "iris (true)", xlab = "PC1", ylab = "PC2")
8    plot(X.prj, col = X.kms.cid + 1, pch = X.kms.cid,
9         main = paste("iris (kmeans)", sprintf("%.4f",
               X.kms.adjR)),
10        xlab = "PC1", ylab = "PC2")
11
12   dev.off()
13 }
```

Plot



Want MBC

for pbdDMAT?

....

http://www.nics.tennessee.edu/getting-an-allocation

Contents

Where to Learn More

1. The **pbdDEMO** package
   http://cran.r-project.org/web/packages/pbdDEMO/
   Vignette: http://goo.gl/eBsIh
2. Our Google Group:
   http://group.r-pbd.org
3. Get an allocation with us!
   http://www.nics.tennessee.edu/getting-an-allocation

Thanks for coming!

Questions? Comments?

Please help us improve this tutorial by completing a short survey:
http://www.surveymonkey.com/s/W8VLJSP