Programming with Big Data in R

George Ostrouchov and Mike Matheson
Oak Ridge National Laboratory

2016 OLCF User Meeting: Day 0 Tutorial
Oak Ridge National Laboratory
Monday, May 23, 2016 Oak Ridge, Tennessee
Why R? Popularity?

IEEE Spectrum’s Ranking of Programming Languages

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>2015 Spectrum Ranking</th>
<th>2014 Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Java</td>
<td>☀️ telefon 📱</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>2. C</td>
<td>☀️ telefon 📱</td>
<td>99.9</td>
<td>99.3</td>
</tr>
<tr>
<td>3. C++</td>
<td>☀️ telefon 📱</td>
<td>99.4</td>
<td>95.5</td>
</tr>
<tr>
<td>4. Python</td>
<td>☀️ telefon 📱</td>
<td>96.5</td>
<td>93.5</td>
</tr>
<tr>
<td>5. C#</td>
<td>☀️ telefon 📱</td>
<td>91.3</td>
<td>92.4</td>
</tr>
<tr>
<td>6. R</td>
<td>☀️ telefon 📱</td>
<td>84.8</td>
<td>84.8</td>
</tr>
<tr>
<td>7. PHP</td>
<td>☀️ telefon 📱</td>
<td>84.5</td>
<td>84.5</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td>☀️ telefon 📱</td>
<td>83.0</td>
<td>78.9</td>
</tr>
<tr>
<td>9. Ruby</td>
<td>☀️ telefon 📱</td>
<td>76.2</td>
<td>74.3</td>
</tr>
<tr>
<td>10. Matlab</td>
<td>☀️ telefon 📱</td>
<td>72.4</td>
<td>72.8</td>
</tr>
</tbody>
</table>

See: [http://spectrum.ieee.org/static/interactive-the-top-programming-languages#index](http://spectrum.ieee.org/static/interactive-the-top-programming-languages#index)
Why R?  Programming with Data


Thanks to Dirk Eddelbuettel for this slide idea and to John Chambers for providing the high-resolution scans of the covers of his books.
<table>
<thead>
<tr>
<th>Why R?</th>
<th>Resources for Learning R</th>
</tr>
</thead>
<tbody>
<tr>
<td>RStudio IDE</td>
<td></td>
</tr>
<tr>
<td>Task Views:</td>
<td></td>
</tr>
<tr>
<td><a href="http://cran.at.r-project.org/web/views">http://cran.at.r-project.org/web/views</a></td>
<td></td>
</tr>
<tr>
<td>Book:</td>
<td></td>
</tr>
<tr>
<td><em>The Art of R Programming</em> by Norm Matloff: <a href="http://nostarch.com/artofr.htm">http://nostarch.com/artofr.htm</a></td>
<td></td>
</tr>
<tr>
<td>Advanced R:</td>
<td></td>
</tr>
<tr>
<td><a href="http://adv-r.had.co.nz/">http://adv-r.had.co.nz/</a> and <a href="http://docs.ggplot2.org/current/">ggplot2</a> by Hadley Wickham</td>
<td></td>
</tr>
<tr>
<td>R programming for those coming from other languages:</td>
<td></td>
</tr>
<tr>
<td>The [R] stackoverflow tag.</td>
<td></td>
</tr>
</tbody>
</table>
Why R?  Programming with Big Data

Engage parallel libraries at scale
R language unchanged
New distributed concepts
New profiling capabilities
New interactive SPMD
In situ distributed capability
In situ staging capability via ADIOS
Plans for DPLASMA GPU capability

pbdr Core Team
Wei-Chen Chen, FDA
George Ostrouchov, ORNL & UTK
Drew Schmidt, UTK

Developers
Christian Heckendorf, Pragneshkumar Patel, Gaurav Sehrawat

Contributors
Whit Armstrong, Ewan Higgs, Michael Lawrence, David Pierce, Brian Ripley, ZhaoKang Wang, Hao Yu
Modules on Titan, Rhea, and Eos (Current R Version is 3.3.0)

Notes - Remember to submit R to compute nodes and not run it on login nodes
Notes - R gpu code can run on Titan nodes or Rhea gpu nodes

```
module load r/3.3.0
R
rstudio (Currently only on Rhea - use either a remote visualization tool or forward X [ssh -X and qsub -X])
```

Example qsub batch script for Titan

```
#!/bin/csh
#PBS -A STF006
#PBS -N R
#PBS -q batch
#PBS -l nodes=1
#PBS -l walltime=0:15:00

cd /lustre/atlas2/stf006/world-shared/mikem
module load r/3.3.0

setenv OPENBLAS_NUM_THREADS 1
setenv OMP_NUM_THREADS 1

echo "host = 'hostname'

aprun -n 1 Rscript --vanilla eigen.r
aprun -n 1 Rscript --vanilla ex_hdf5.r
aprun -n 1 Rscript --vanilla ex_max.r
```
Strategies for Making R, a Scripting Language, Faster

Serial solutions before parallel solutions

- User R code often inefficient (high-level code = deep complexity)
  - Profile and improve code first
  - Vectorize loops if possible
  - Compute once if not changing
  - Know when copies are made
- Move kernels into compiled language, such as C/C++ (+OpenMP)
- **multicore** components of **parallel** package (Unix fork)
- Distributed via **pbdR** (only solution for big memory)
Integrating C/C++ Code Into R

**.Call**
- Standard R interface to C code
- Lightweight but clunky

**Rcpp: Incorporating C++ code into R**
Authors: Dirk Eddelbuettel and Romain Francois
- Simplifies integrating C++ code with R
- Maps R objects (vectors, matrices, functions, environments, ...) to dedicated C++ classes
- Broad support for C++ Standard Template Library idioms.
- C++ code can be compiled, linked and loaded on the fly, or added via packages.
- Error and exception code handling
Rcpp Example: A simple row max calculation

```cpp
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]

NumericVector row_max(NumericMatrix m)
{
  int nrows = m.nrow();
  NumericVector maxPerRow(nrows);

  for (int i = 0; i < nrows; i++)
  {
    maxPerRow[i] = Rcpp::max(m(i, _));
  }

  return (maxPerRow);
}
```

One can get configuration values by

```bash
setenv PKG_CXXFLAGS `Rscript -e "Rcpp::CxxFlags()"`
setenv PKG_LIBS `Rscript -e "Rcpp::LdFlags()"`
```
### Rcpp Example (con’d): A simple row max calculation

```r
library( Rcpp )
Sys.setenv("PKG_CXXFLAGS" = 
  "-I/sw/redhat6/r/3.3.0/rhel6_gnu4.8.2/lib64/R/library/Rcpp/include" )
Sys.setenv("PKG_LIBS"="-lm"
)
sourceCpp( "ex_max.cpp" )

set.seed( 27 )
X <- matrix( rnorm( 4 * 4 ), 4, 4 )
X

print( "Rcpp" )
row_max( X )
```

#### Rscript ex_max.r

```r
Rscript ex_max.r

[1,] 1.9071626 -1.093468881 2.134637891 1.144876890
[2,] 1.1448769 0.295241218 0.237844610 0.006885942
[3,] -0.7645307 0.006885942 -1.285127360 -0.7457995
[4,] -1.4574325 1.157410886 0.034827250 -1.0688030

[1] "Rcpp"
[1] 2.134637891 1.144876890 0.006885942 1.157410886
```
The RcppArmadillo package is a set of bindings to the Armadillo C++ library.

Armadillo is a templated C++ linear algebra library that uses supplied BLAS and LAPACK.

Includes some machine learning libraries

BLAS and LAPACK are also directly engaged from R.

Probably not faster than R direct but not having to come back out to R if C++ code needs to use linear algebra can produce gains.

RcppArmadillo Example: Eigenvalue calculation

```cpp
cat eigen.cpp
#include <RcppArmadillo.h>
//[[Rcpp::depends(RcppArmadillo)]]
//[[Rcpp::export]]
arma::vec getEigenValues( arma::mat M )
{
  return ( arma::eig_sym( M ) );
}
```
RcppArmadillo Example (con’d): Eigenvalue calculation

```r
library( Rcpp )
library( RcppArmadillo )
Sys.setenv( "PKG_CXXFLAGS" = "-I/sw/redhat6/r/3.3.0/rhel6_gnu4.8.2/lib64/R/library/RcppArmadillo/include" )
Sys.setenv( "PKG_LIBS"="-lm" )
sourceCpp( "eigen.cpp" )

set.seed( 27 )
X <- matrix( rnorm( 4 * 4 ), 4, 4 )
Z <- X %*% t( X )
print( "RcppArmadillo" )
getEigenValues( Z )
print( "R" )
eigen( Z )$values
```

Rscript eigen.r

```
[1] "RcppArmadillo"
[1,]
[1,] 0.03779289
[2,] 0.85043786
[3,] 2.03877658
[4,] 17.80747601
[1] "R"
[1] 17.80747601 2.03877658 0.85043786 0.03779289
```
I/O Packages

- **function fread in package *data.table***: fast and easy csv
- **rhdf5**: fast and easy HDF5 I/O
- **pbdNCDF4**: fast NetCDF4 collective read and write
- **pbdADIOS** (on GitHub, under development): fast bp I/O with ADIOS staging capability
- **pbdIO** (on GitHub, under development): Easy parallel I/O, includes parallel csv with load balance

Parallel chunking: Read the most natural way from disk

- C: by blocks of rows
- FORTRAN: by blocks of columns
- CSV best with groups of files
- Parallel best with binary, fixed format
### rhdf5 Example: Write and then read a matrix

```r
library( rhdf5 )
print( "Writing hdf5" )
h5createFile( "test.h5" )
h5createGroup( "test.h5", "MainGroup" )
X <- matrix( rnorm( 3 * 3 ), ncol = 3, nrow = 3 )
X
h5write( X, file = "test.h5", "MainGroup/Matrix", write.attributes = FALSE )
h5ls( "test.h5" )
print( "Reading hdf5" )
Y <- h5read( "test.h5", "/MainGroup/Matrix" )
Y
```

### Loading required `package: methods`

```r
[1] "Writing hdf5"
[1] TRUE
[1] TRUE
[,1] [,2] [,3]
[1,] 0.9124038 1.0390048 -1.1731370
[2,] -0.8973774 0.3447025 -0.1201449
[3,] 1.6489298 -0.1993730 1.1330055

group name otype dclass dim
0 / MainGroup H5I_GROUP
1 /MainGroup Matrix H5I_DATASET FLOAT 3 x 3

[1] "Reading hdf5"

[,1] [,2] [,3]
[1,] 0.9124038 1.0390048 -1.1731370
[2,] -0.8973774 0.3447025 -0.1201449
[3,] 1.6489298 -0.1993730 1.1330055
```
rhdf5 Example (con’d): Check file contents outside of R

h5dump test.h5

```
HDF5 "test.h5" {
  GROUP "/" {
    GROUP "MainGroup" {
      DATASET "Matrix" {
        DATATYPE H5T_IEEE_F64LE
        DATASPACE SIMPLE { ( 3, 3 ) / ( 3, 3 ) }
        DATA {
          (0,0): 0.912404, -0.897377, 1.64893,
          (1,0): 1.039, 0.344703, -0.199373,
          (2,0): -1.17314, -0.120145, 1.13301
        }
      }
    }
  }
}
```

Note: rhdf5 enables reading chunks and slabs of HDF5 file arrays in R for fast parallel reads from the lustre file system.
Why R? Programming with Big Data

**pbdr Core Team**
Wei-Chen Chen, FDA
George Ostrouchov, ORNL & UTK
Drew Schmidt, UTK

**Developers**
Christian Heckendorf, Pragneshkumar Patel, Gaurav Sehrawat

**Contributors**
Whit Armstrong, Ewan Higgs, Michael Lawrence, David Pierce, Brian Ripley, ZhaoKang Wang, Hao Yu

- Engage parallel libraries at scale
- R language unchanged
- New distributed concepts
- New profiling capabilities
- New interactive SPMD
- In situ distributed capability
- In situ staging capability via ADIOS
- Plans for DPLASMA GPU capability
HPC Cluster with NVRAM and Parallel File System

Today's HPC Cluster

Parallel File System

Disk

Servers

Compute Nodes

I/O Nodes

Login Nodes

Your Laptop

Big Data

"Little Data"

Solid State Disk

Multicore

Programming with Big Data in R
Why use HPC libraries?

- Many science communities are invested in their API.
- Data analysis uses much of the same basic math as simulation science.
- The libraries represent 30+ years of parallel algorithm research.
- *They’re tested. They’re fast. They’re scalable.*
pbdMPI: a High Level Interface to MPI

- API is simplified: defaults in control objects.
- S4 methods: extensible to complex R objects.
- Additional error checking
- Array and matrix methods without serialization: faster than Rmpi.

<table>
<thead>
<tr>
<th>pbdMPI (S4)</th>
<th>Rmpi</th>
</tr>
</thead>
<tbody>
<tr>
<td>allreduce</td>
<td>mpi.allreduce</td>
</tr>
<tr>
<td>allgather</td>
<td>mpi.allgather, mpi.allgatherv, mpi.allgather.Robj</td>
</tr>
<tr>
<td>bcast</td>
<td>mpi.bcast, mpi.bcast.Robj</td>
</tr>
<tr>
<td>gather</td>
<td>mpi.gather, mpi.gatherv, mpi.gather.Robj</td>
</tr>
<tr>
<td>recv</td>
<td>mpi.recv, mpi.recv.Robj</td>
</tr>
<tr>
<td>reduce</td>
<td>mpi.reduce</td>
</tr>
<tr>
<td>scatter</td>
<td>mpi.scatter, mpi.scatterv, mpi.scatter.Robj</td>
</tr>
<tr>
<td>send</td>
<td>mpi.send, mpi.send.Robj</td>
</tr>
</tbody>
</table>
A simple SPMD allreduce

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

## Your local computation
n <- comm.rank() + 1

## Now "Reduce" and give the result to all
all_sum <- allreduce(n) # Sum is default

text <- paste("Hello: n is", n, "sum is", all_sum )
comm.print(text, all.rank=TRUE)

finalize()
```

Execute this batch script via:

```
mpirun -np 2 Rscript allreduce.r
```

Output:

```
COMM.RANK = 0
[1] "Hello: n is 1 sum is 3"
COMM.RANK = 1
[1] "Hello: n is 2 sum is 3"
```
Example: Letter Recognition data from package `mlbench` (20,000 × 17)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1 | [,1] | lettr | capital letter |
| 2 | [,2] | x.box | horizontal position of box |
| 3 | [,3] | y.box | vertical position of box |
| 4 | [,4] | width | width of box |
| 5 | [,5] | high | height of box |
| 6 | [,6] | onpix | total number of on pixels |
| 7 | [,7] | x.bar | mean x of on pixels in box |
| 8 | [,8] | y.bar | mean y of on pixels in box |
| 9 | [,9] | x2bar | mean x variance |
|10 | [,10] | y2bar | mean y variance |
|11 | [,11] | xybar | mean x y correlation |
|12 | [,12] | x2ybr | mean of x^2 y |
|13 | [,13] | xy2br | mean of x y^2 |
|14 | [,14] | x.ege | mean edge count left to right |
|15 | [,15] | xegvy | correlation of x.ege with y |
|16 | [,16] | y.ege | mean edge count bottom to top |
|17 | [,17] | yegvx | correlation of y.ege with x |

Example: Random Forest Code
(build many simple models from subsets, use model averaging to predict)

Serial Code 4_rf_s.r

```r
library(randomForest)
library(mlbench)
data(LetterRecognition) # 26 Capital Letters Data 20,000 x 17
set.seed(seed=123)
n <- nrow(LetterRecognition)
n_test <- floor(0.2*n)
i_test <- sample.int(n, n_test) # Use 1/5 of the data to test
train <- LetterRecognition[-i_test, ]
test <- LetterRecognition[i_test, ]

## train random forest
rf.all <- randomForest(lettr ~ ., train, ntree=500, norm.votes=FALSE)

## predict test data
pred <- predict(rf.all, test)
correct <- sum(pred == test$lettr)
cat("Proportion Correct:", correct/(n_test), "\n")
```
Example: Random Forest Code
(Split learning by blocks of trees. Split prediction by blocks of rows.)

Parallel Code 4_r_f_p.r

```
library(randomForest)
library(mlbench)
data(LetterRecognition)
comm.set.seed(seed=123, diff=FALSE)  # same training data
n <- nrow(LetterRecognition)
n_test <- floor(0.2*n)
i_test <- sample.int(n, n_test)  # Use 1/5 of the data to test
train <- LetterRecognition[-i_test, ]
test <- LetterRecognition[i_test, ][get.jid(n_test), ]

comm.set.seed(seed=1e6*runif(1), diff=TRUE)
my.rf <- randomForest(lettr ~ ., train, ntree=500/%comm.size(), norm.votes=FALSE)
rf.all <- do.call(combine, allgather(my.rf))
pred <- predict(rf.all, test)
correct <- allreduce(sum(pred == test$lettr))

comm.cat("Proportion Correct:", correct/(n_test), "\n")
```
A matrix is mapped to a processor grid shape:

(a) $1 \times 6$

\[
\begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 \\
3 & 4 & 5 \\
\end{bmatrix}
\]

(b) $2 \times 3$

\[
\begin{bmatrix}
0 & 1 & 2 \\
2 & 3 \\
4 & 5 \\
\end{bmatrix}
\]

(c) $3 \times 2$

\[
\begin{bmatrix}
0 & 1 \\
2 & 3 \\
4 & 5 \\
\end{bmatrix}
\]

(d) $6 \times 1$

\[
\begin{bmatrix}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{bmatrix}
\]

Table: Processor Grid Shapes with 6 Processors
Distributed Matrix and Vector Operations

**Powered by ScaLAPACK, PBLAS, and BLACS (MKL, SciLIB, or ACML)**

- Block-cyclic data layout for scalability and efficiency
- No change in R syntax
- High-level convenience for data layout redistributions
  - Row-major data: read row-block then convert to block-cyclic
  - Column-major data: read column-block then convert to block-cyclic

**Global and local views of block-cyclic on a $2 \times 3$ processor grid**
No change in syntax. Data redistribution functions.

```
x <- x[-1, 2:5]
x <- log(abs(x) + 1)
x.pca <- prcomp(x)
xtx <- t(x) %*% x
ans <- svd(solve(xtx))
```

The above (and over 100 other functions) runs on 1 core with R or 10,000 cores with `pbdR ddmatrix` class.

```
> showClass("ddmatrix")
Class "ddmatrix" [package "pbdDMAT"]
Slots:
Name: Data dim ldim bldim ICTXT
Class: matrix numeric numeric numeric numeric
```

```
> x <- as.rowblock(x)
> x <- as.colblock(x)
> x <- redistribute(x, bldim=c(8, 8), ICTXT = 0)
```
Truncated SVD from random projections\(^1\)

**Prototype for Randomized SVD**

Given an \( m \times n \) matrix \( A \), a target number \( k \) of singular vectors, and an exponent \( q \) (say, \( q = 1 \) or \( q = 2 \)), this procedure computes an approximate rank-\( 2k \) factorization \( U \Sigma V^* \), where \( U \) and \( V \) are orthonormal, and \( \Sigma \) is nonnegative and diagonal.

**Stage A:**
1. Generate an \( n \times 2k \) Gaussian test matrix \( \Omega \).
2. Form \( Y = (AA^*)^q A \Omega \) by multiplying alternately with \( A \) and \( A^* \).
3. Construct a matrix \( Q \) whose columns form an orthonormal basis for the range of \( Y \).

**Stage B:**
4. Form \( B = Q^* A \).
5. Compute an SVD of the small matrix: \( B = \tilde{U} \Sigma \tilde{V}^* \).
6. Set \( U = Q \tilde{U} \).

**Note:** The computation of \( Y \) in step 2 is vulnerable to round-off errors. When high accuracy is required, we must incorporate an orthonormalization step between each application of \( A \) and \( A^* \); see Algorithm 4.4.

**Algorithm 4.4: Randomized Subspace Iteration**

Given an \( m \times n \) matrix \( A \) and integers \( \ell \) and \( q \), this algorithm computes an \( m \times \ell \) orthonormal matrix \( Q \) whose range approximates the range of \( A \).

1. Draw an \( n \times \ell \) standard Gaussian matrix \( \Omega \).
2. Form \( Y_0 = A \Omega \) and compute its QR factorization \( Y_0 = Q_0 R_0 \).
3. for \( j = 1, 2, \ldots, q \) 
4. Form \( \bar{Y}_j = A^* Q_{j-1} \) and compute its QR factorization \( \bar{Y}_j = Q_j \bar{R}_j \).
5. Form \( Y_j = A \bar{Q}_j \) and compute its QR factorization \( Y_j = Q_j R_j \).
6. end
7. \( Q = Q_q \).


---

Serial R

rSVD <- function(A, k, q=3) 
{
  ## Stage A
  Omega <- matrix(rnorm(n*2*k), nrow=n, ncol=2*k)
  Y <- A Omega
  Q <- qr.Q(qr(Y))
  At <- t(A)
  for(i in 1:q)
    { Y <- At %*% Q
      Q <- qr.Q(qr(Y))
      Y <- A %*% Q
      Q <- qr.Q(qr(Y))
    }
  ## Stage B
  B <- t(Q) %*% A
  U <- La.svd(B)$u
  U <- Q %*% U
  U[, 1:k]
}
Truncated SVD from random projections

Serial R

```r
rSVD <- function(A, k, q=3) {
    ## Stage A
    Omega <- matrix(rnorm(n*2*k), nrow=n, ncol=2*k)
    Y <- A %*% Omega
    Q <- qr.Q(qr(Y))
    At <- t(A)
    for (i in 1:q) {
        Y <- At %*% Q
        Q <- qr.Q(qr(Y))
    }

    ## Stage B
    B <- t(Q) %*% A
    U <- La.svd(B)$u
    U <- Q %*% U
    U[, 1:k]
}
```

Parallel pbdR

```r
rSVD <- function(A, k, q=3) {
    ## Stage A
    Omega <- ddmatrix("rnorm", nrow=n, ncol=2*k)
    Y <- A %*% Omega
    Q <- qr.Q(qr(Y))
    At <- t(A)
    for (i in 1:q) {
        Y <- At %*% Q
        Q <- qr.Q(qr(Y))
    }

    ## Stage B
    B <- t(Q) %*% A
    U <- La.svd(B)$u
    U <- Q %*% U
    U[, 1:k]
}
```
From journal to scalable code and scaling data in one day.

30 Singular Vectors from a 100,000 by 1,000 Matrix

Algorithm
- full
- randomized

Speedup relative to 1 core

rSVD speedup relative to full SVD

Programming with Big Data in R
RSVD vs SVD Performance

<table>
<thead>
<tr>
<th>host: eos</th>
<th>host: rhea</th>
<th>host: titan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Method**: rsvd, svd
- **Time (sec)**
- **Number of Nodes**

Programming with Big Data in R
Benchmarking (134 GB) SVD Calculations with pbdR

RSVD vs SVD Performance

- host: eos
- host: rhea
- host: titan

Number of Cores

Time (sec)

as.factor(npernode)

• 2
• 16

method

- rsvd
- svd

Programming with Big Data in R
suppressMessages(library(rhdf5))
suppressMessages(library(pbdDMAT, quiet=TRUE))
suppressMessages(library(pbdML, quiet=TRUE))

start.time = Sys.time()
init.grid()
end.time = Sys.time()
barrier()

comm.print(paste("initgrid = ", end.time - start.time))

args = commandArgs(trailingOnly = TRUE)
meth = args[1]
npernode = strtoi(args[2])
n_keep = strtoi(args[3])
block_row = strtoi(args[4])
block_col = strtoi(args[5])

nproc <- comm.size()

rows <- 12390000 /\!/ nproc
cols <- 1250

len <- rows*cols*4
start <- comm.rank() * len

## this one has individual files

if (nproc == 2 ) fn <- paste("X/X2", comm.rank(), "h5", sep=".")
if (nproc == 4 ) fn <- paste("X/X4", comm.rank(), "h5", sep=".")
if (nproc == 5 ) fn <- paste("X/X5", comm.rank(), "h5", sep=".")
if (nproc == 10) fn <- paste("X/X10", comm.rank(), "h5", sep=".")
if (nproc == 20) fn <- paste("X/X20", comm.rank(), "h5", sep=".")
if (nproc == 100) fn <- paste("X/X100", comm.rank(), "h5", sep=".")
if (nproc == 200) fn <- paste("X/X200", comm.rank(), "h5", sep=".")
if ( nproc == 400 ) fn <- paste("X/X400", comm.rank(), "h5", sep=".")
if ( nproc == 600 ) fn <- paste("X/X600", comm.rank(), "h5", sep=".")
if ( nproc == 1000 ) fn <- paste("X/X1000", comm.rank(), "h5", sep=".")
if ( nproc == 2000 ) fn <- paste("X3/X2000", comm.rank(), "h5", sep=".")
if ( nproc == 3000 ) fn <- paste("X3/X3000", comm.rank(), "h5", sep=".")
if ( nproc == 5000 ) fn <- paste("X5/X5000", comm.rank(), "h5", sep=".")
if ( nproc == 30000 ) fn <- paste("X/X30000", comm.rank(), "h5", sep=".")

start.time = Sys.time()
A <- h5read(fn, "/dataset" )
end.time = Sys.time()
barrier()
comm.print( paste("io = ", end.time - start.time ) )

## comm.print( A[ 1:5 , 1:5 ], all.rank = TRUE )
## comm.print( dim( A ), all.rank = TRUE )

start.time = Sys.time()
A <- new("ddmatrix", Data=A, dim=c(12390000, 1250), ldim=dim(A), bldim=dim(A), ICTXT=2)

## comm.print( dim(submatrix( A )), all.rank = TRUE )
## comm.print( submatrix( A )[ 1:5 , 1:5 ], all.rank = TRUE )
## comm.print( A, all.rank = TRUE )

A <- as.blockcyclic( A, bldim = c( block_row, block_col ) )

## comm.print( A[ 1:5 , 1:5 ], all.rank = TRUE )
## comm.print( dim( A ), all.rank = TRUE )

end.time = Sys.time()
barrier()
comm.print( paste("blockcyclic = ", end.time - start.time ) )

## comm.print( A, all.rank = TRUE )
## comm.print( submatrix( A )[ 1:5 , 1:5 ], all.rank = TRUE )
Benchmarking (134 GB) SVD Calculations with \texttt{pbdR}

```r
require(pbdR)

meth <- "rsvd"
start.time = Sys.time()
if ( meth == "rsvd" )
  Res <- rsvd( A, k = n_keep, q = 3, retu = TRUE, retv = TRUE )
if ( meth == "gpu_rsvd" )
  Res <- rsvd( A, k = n_keep, q = 3, retu = TRUE, retv = TRUE )
if ( meth == "gpu_svd" )
  Res <- svd( A, nu = n_keep, nv = n_keep )
if ( meth == "svd" )
  Res <- svd( A, nu = n_keep, nv = n_keep )
end.time = Sys.time()
barrier()
comm.print( paste( "compute = ", end.time - start.time ) )
comm.print( Res$d )
mesg <- paste( "Finished ... method =", meth, "nproc =", nproc, "npernode =", npernode, "keep =", n_keep, "blocking =", block_row )
comm.print( mesg )
finalize()
```
Future Work

Where to learn more?

- [http://r-pbd.org/](http://r-pbd.org/)
- pbdDEMO vignette
- [Googlegroup:RBigDataProgramming](https://groups.google.com)
- pbdR Installations: OLCF, NERSC, SDSC, TACC, IU, BSC Spain, CSCS Switzerland, IT4I Czech, ISM Japan, and many more

Support

This work 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 material is based upon work supported by the National Science Foundation Division of Mathematical Sciences under Grant No. 1418195.

This work also used resources of National Institute for Computational Sciences at the University of Tennessee, Knoxville, which is supported by the U.S. National Science Foundation.