Programming with Big Data in R

George Ostrouchov and Mike Matheson

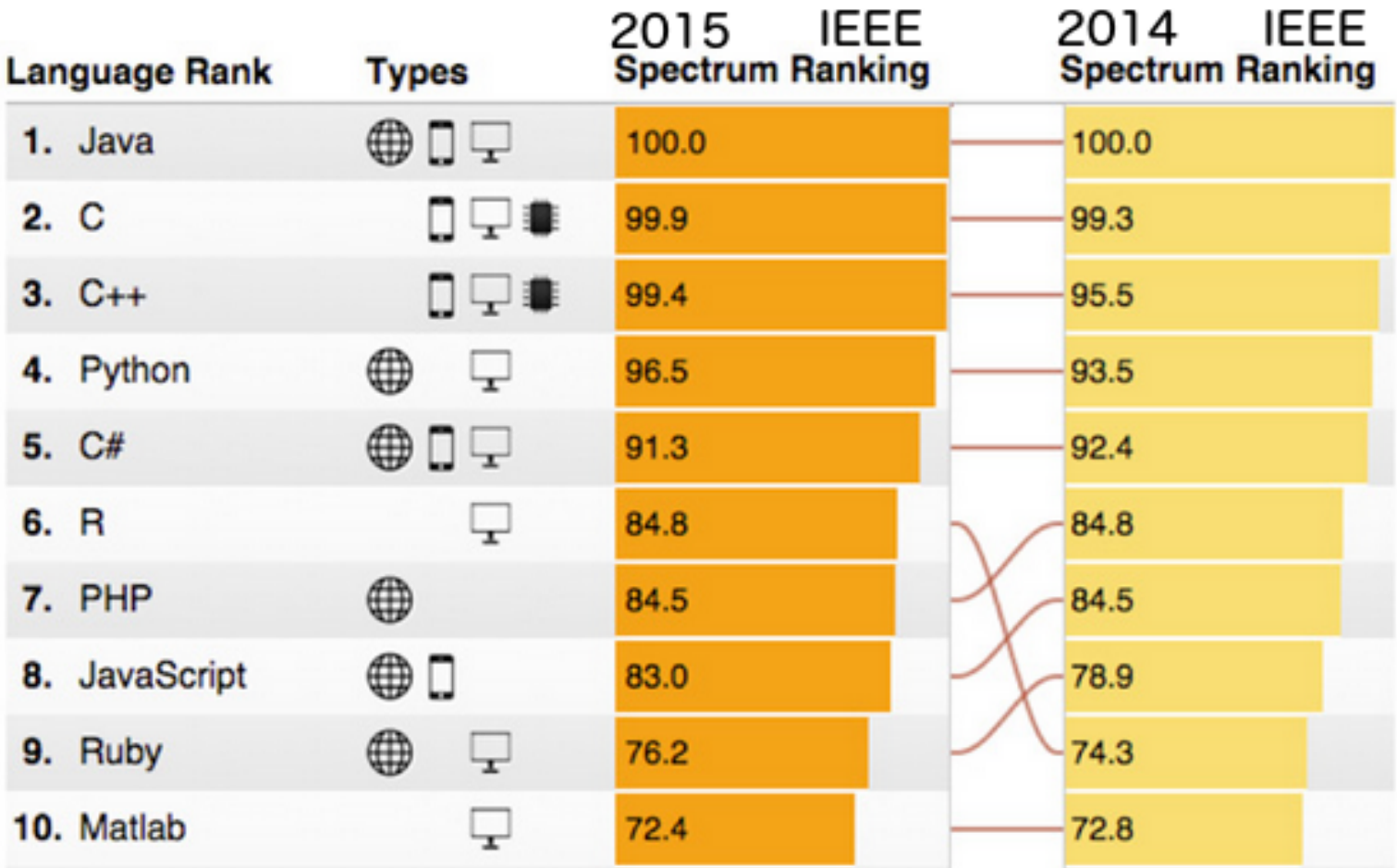
Oak Ridge National Laboratory

2016 OLCF User Meeting: Day 0 Tutorial

Oak Ridge National Laboratory
Monday, May 23, 2016 Oak Ridge, Tennessee

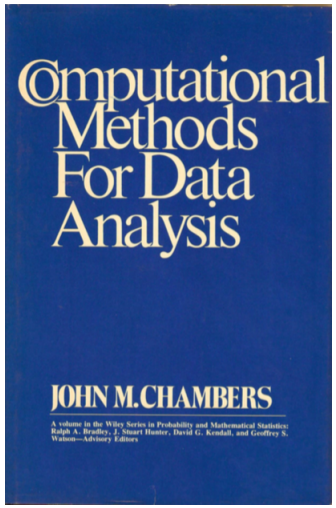
Why R? Popularity?

IEEE Spectrum's Ranking of Programming Languages

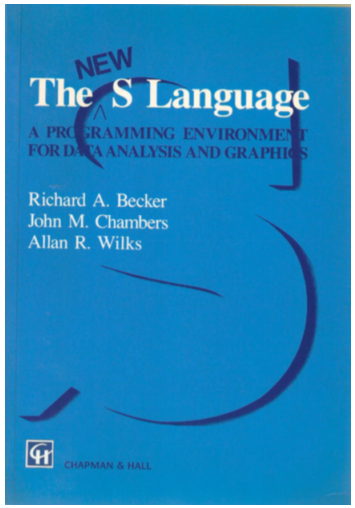


See: http://spectrum.ieee.org/static/interactive-the-top-programming-languages#index

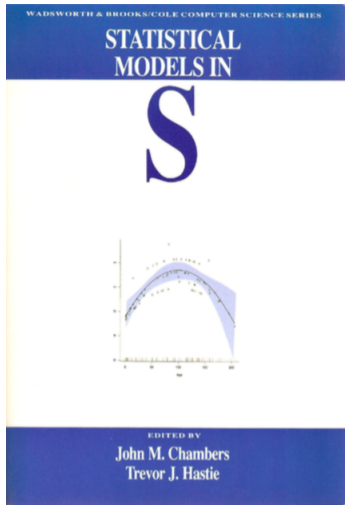
Why R? Programming with Data



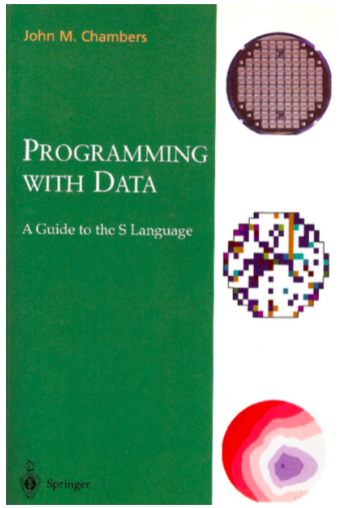
Chambers. *Computational Methods for Data Analysis*. Wiley, 1977.



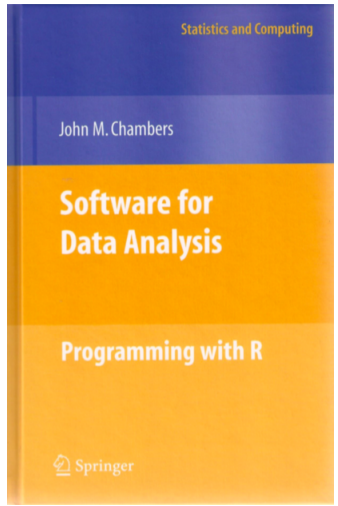
Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.



Chambers and Hastie. *Statistical Models in S*. Chapman & Hall, 1992.



Chambers. *Programming with Data*. Springer, 1998.



Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008.

Thanks to Dirk Eddelbuettel for this slide idea and to John Chambers for providing the high-resolution scans of the covers of his books.

Why R? Resources for Learning R

- RStudio IDE
  http://www.rstudio.com/products/rstudio-desktop/
- *Task Views*: http://cran.at.r-project.org/web/views
- Book: *The Art of R Programming* by Norm Matloff:
  http://nostarch.com/artofr.htm
- *Advanced R*: http://adv-r.had.co.nz/ and *ggplot2*
  http://docs.ggplot2.org/current/ by Hadley Wickham
- R programming for those coming from other languages: http://www.johndcook.com/R_language_for_programmers.html
- *aRrgh: a newcomer's (angry) guide to R*, by Tim Smith and Kevin Ushey: http://tim-smith.us/arrgh/
- Mailing list archives: http://tolstoy.newcastle.edu.au/R/
- The [R] stackoverflow tag.

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

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

cat ex_max.cpp

```
1  #include <Rcpp.h>
2  using namespace Rcpp;
3
4  //[[Rcpp::export]]
5
6  NumericVector row_max( NumericMatrix m )
7  {
8     int nrows = m.nrow( );
9     NumericVector maxPerRow( nrows );
10
11    for ( int i = 0; i < nrows; i++ )
12    {
13       maxPerRow[ i ] = Rcpp::max( m( i, _ ) );
14    }
15
16    return ( maxPerRow );
17 }
```

One can get configuration values by

```
1  setenv PKG_CXXFLAGS  `Rscript -e "Rcpp:::CxxFlags( )"`
2  setenv PKG_LIBS `Rscript -e "Rcpp:::LdFlags( )"`
```

Rcpp Example (con'd): A simple row max calculation

cat ex_max.r

```
1  library( Rcpp )
2  Sys.setenv( "PKG_CXXFLAGS" =
3   "-I /sw/redhat6/r/3.3.0/rhel6_gnu4.8.2/lib64/R/library/Rcpp/include" )
4  Sys.setenv( "PKG_LIBS"="-lm" )
5
6  sourceCpp( "ex_max.cpp" )
7
8  set.seed( 27 )
9  X <- matrix( rnorm( 4 * 4 ), 4, 4 )
10 X
11
12 print( "Rcpp" )
13 row_max( X )
```

Rscript ex_max.r

```
1  Rscript ex_max.r
2              [,1]          [,2]         [,3]        [,4]
3  [1,]  1.9071626 -1.093468881  2.13463789  1.5702953
4  [2,]  1.1448769  0.295241218  0.23784461  0.1580101
5  [3,] -0.7645307  0.006885942 -1.28512736 -0.7457995
6  [4,] -1.4574325  1.157410886  0.03482725 -1.0688030
7  [1] "Rcpp"
8  [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

cat eigen.cpp

```
1 #include <RcppArmadillo.h>
2 //[[Rcpp::depends(RcppArmadillo)]]
3 //[[Rcpp::export]]
4
5 arma::vec getEigenValues( arma::mat M )
6 {
7     return ( arma::eig_sym( M ) );
8 }
```

RcppArmadillo Example (con'd): Eigenvalue calculation

cat eigen.r

```
1  library( Rcpp )
2  library( RcppArmadillo )
3  Sys.setenv( "PKG_CXXFLAGS" = "-I
       /sw/redhat6/r/3.3.0/rhel6_gnu4.8.2/lib64/R/library/RcppArmadillo/include" )
4  Sys.setenv( "PKG_LIBS"="-lm"
5  )
6  sourceCpp( "eigen.cpp" )
7
8  set.seed( 27 )
9  X <- matrix( rnorm( 4 * 4 ), 4, 4 )
10 Z <- X %*% t( X )
11 print( "RcppArmadillo" )
12 getEigenValues( Z )
13
14 print( "R" )
15 eigen( Z )$values
```

Rscript eigen.r

```
1  [1] "RcppArmadillo"
2               [,1]
3  [1,]  0.03779289
4  [2,]  0.85043786
5  [3,]  2.03877658
6  [4,] 17.80747601
7  [1] "R"
8  [1] 17.80747601  2.03877658  0.85043786  0.03779289
```

I/O

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 developent): 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

cat wr_hdf5.r

```
1  library( rhdf5 )
2  print( "Writing hdf5" )
3  h5createFile( "test.h5" )
4  h5createGroup( "test.h5", "MainGroup" )
5  X <- matrix( rnorm( 3 * 3 ), ncol = 3, nrow = 3 )
6  X
7  h5write( X, file = "test.h5", "MainGroup/Matrix", write.attributes = FALSE )
8  h5ls( "test.h5" )
9  print( "Reading hdf5" )
10 Y <- h5read( "test.h5", "/MainGroup/Matrix" )
11 Y
```

Rscript wr_hdf5.r

```
1  Loading required package: methods
2  [1] "Writing hdf5"
3  [1] TRUE
4  [1] TRUE
5             [,1]       [,2]       [,3]
6  [1,]  0.9124038  1.0390048 -1.1731370
7  [2,] -0.8973774  0.3447025 -0.1201449
8  [3,]  1.6489298 -0.1993730  1.1330055
9        group       name        otype dclass   dim
10 0         / MainGroup   H5I_GROUP
11 1 /MainGroup     Matrix H5I_DATASET  FLOAT 3 x 3
12 [1] "Reading hdf5"
13            [,1]       [,2]       [,3]
14 [1,]  0.9124038  1.0390048 -1.1731370
15 [2,] -0.8973774  0.3447025 -0.1201449
16 [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

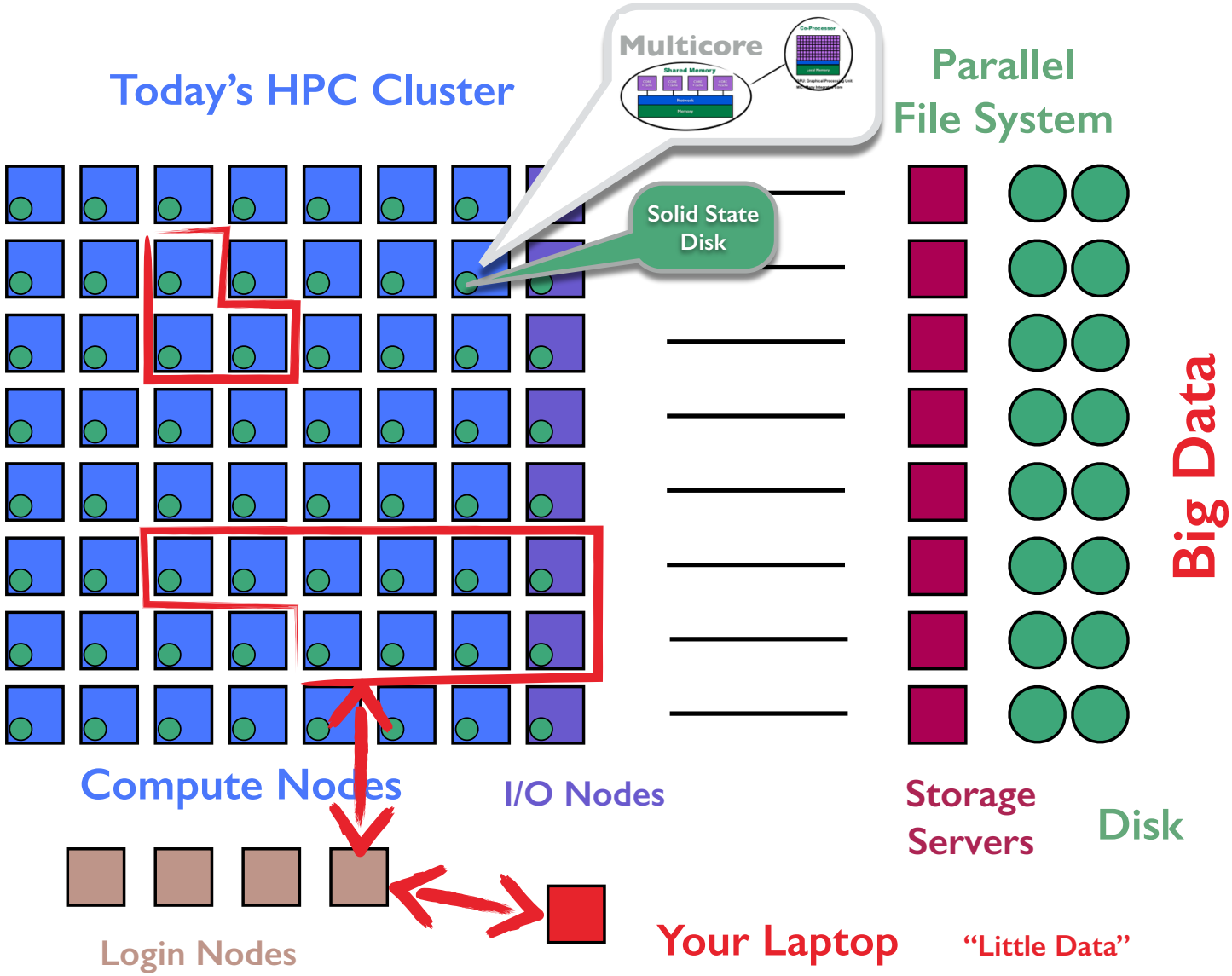
pbdR Core Team
Wei-Chen Chen, FDA
George Ostrouchov, ORNL & UTK
Drew Schmidt, UTK

Developers
Christian Heckendorf, Pragneshkumar Patel, Gaurav Sehrawat

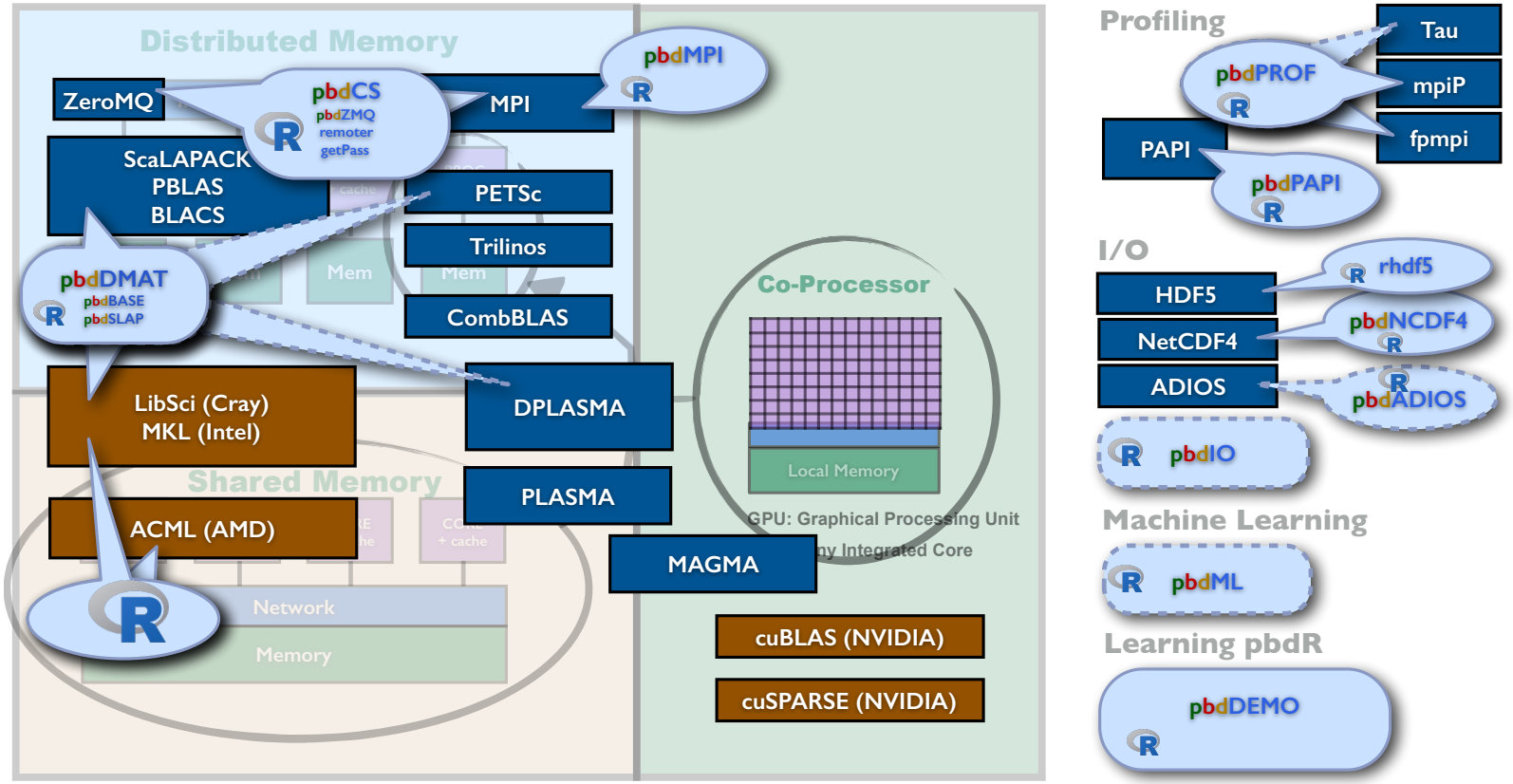
Contributors
Whit Armstrong, Ewan Higgs, Michael Lawrence, David Pierce, Brian Ripley, ZhaoKang Wang, Hao Yu

- 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



pbdR Interfaces to Libraries: Sustainable Path



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**.

pbdMPI (S4)	**Rmpi**
allreduce	mpi.allreduce
allgather	mpi.allgather, mpi.allgatherv, mpi.allgather.Robj
bcast	mpi.bcast, mpi.bcast.Robj
gather	mpi.gather, mpi.gatherv, mpi.gather.Robj
recv	mpi.recv, mpi.recv.Robj
reduce	mpi.reduce
scatter	mpi.scatter, mpi.scatterv, mpi.scatter.Robj
send	mpi.send, mpi.send.Robj

SPMD: Copies of One Code Run Asynchronously

A simple SPMD `allreduce`

allreduce.r

```
1  library(pbdMPI, quiet = TRUE)
2
3  ## Your local computation
4  n <- comm.rank() + 1
5
6  ## Now "Reduce" and give the result to all
7  all_sum <- allreduce(n) # Sum is default
8
9  text <- paste("Hello: n is", n, "sum is", all_sum )
10 comm.print(text, all.rank=TRUE)
11
12 finalize()
```

Execute this batch script via:

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

Output:

```
1 COMM.RANK = 0
2 [1] "Hello: n is 1 sum is 3"
3 COMM.RANK = 1
4 [1] "Hello: n is 2 sum is 3"
```

Machine Learning Example: Random Forest

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
```

P. W. Frey and D. J. Slate (Machine Learning Vol 6/2 March 91): "Letter Recognition Using Holland-style Adaptive Classifiers".

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

Serial Code 4_rf_s.r

```
1  library(randomForest)
2  library(mlbench)
3  data(LetterRecognition) # 26 Capital Letters Data 20,000 x 17
4  set.seed(seed=123)
5  n <- nrow(LetterRecognition)
6  n_test <- floor(0.2*n)
7  i_test <- sample.int(n, n_test) # Use 1/5 of the data to test
8  train <- LetterRecognition[-i_test, ]
9  test <- LetterRecognition[i_test, ]
10
11 ## train random forest
12 rf.all <- randomForest(lettr ~ ., train, ntree=500, norm.votes=FALSE)
13
14 ## predict test data
15 pred <- predict(rf.all, test)
16 correct <- sum(pred == test$lettr)
17 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_rf_p.r

```
1  library(randomForest)
2  library(mlbench)
3  data(LetterRecognition)
4  comm.set.seed(seed=123, diff=FALSE) # same training data
5  n <- nrow(LetterRecognition)
6  n_test <- floor(0.2*n)
7  i_test <- sample.int(n, n_test) # Use 1/5 of the data to test
8  train <- LetterRecognition[-i_test, ]
9  test <- LetterRecognition[i_test, ][get.jid(n_test), ]
10
11 comm.set.seed(seed=1e6*runif(1), diff=TRUE)
12 my.rf <- randomForest(lettr ~ ., train, ntree=500%/%comm.size(), norm.votes=FALSE)
13 rf.all <- do.call(combine, allgather(my.rf))
14
15 pred <- predict(rf.all, test)
16 correct <- allreduce(sum(pred == test$lettr))
17 comm.cat("Proportion Correct:", correct/(n_test), "\n")
```

Distributed Matrix and Vector Operations

A matrix is mapped to a processor grid shape

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

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

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

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

(a) 1×6 (b) 2×3 (c) 3×2 (d) 6×1

Table: Processor Grid Shapes with 6 Processors

Distributed Matrix and Vector Operations pbdDMAT

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 2×3 processor grid

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

$$\left[\begin{array}{cc|cc} x_{11} & x_{12} & x_{17} & x_{18} \\ x_{21} & x_{22} & x_{27} & x_{28} \\ \hline x_{51} & x_{52} & x_{57} & x_{58} \\ x_{61} & x_{62} & x_{67} & x_{68} \\ \hline x_{91} & x_{92} & x_{97} & x_{98} \end{array}\right]_{5\times 4} \left[\begin{array}{cc|c} x_{13} & x_{14} & x_{19} \\ x_{23} & x_{24} & x_{29} \\ \hline x_{53} & x_{54} & x_{59} \\ x_{63} & x_{64} & x_{69} \\ \hline x_{93} & x_{94} & x_{99} \end{array}\right]_{5\times 3} \left[\begin{array}{cc} x_{15} & x_{16} \\ x_{25} & x_{26} \\ \hline x_{55} & x_{56} \\ x_{65} & x_{66} \\ \hline x_{95} & x_{96} \end{array}\right]_{5\times 2}$$

$$\left[\begin{array}{cc|cc} x_{31} & x_{32} & x_{37} & x_{38} \\ x_{41} & x_{42} & x_{47} & x_{48} \\ \hline x_{71} & x_{72} & x_{77} & x_{78} \\ x_{81} & x_{82} & x_{87} & x_{88} \end{array}\right]_{4\times 4} \left[\begin{array}{cc|c} x_{33} & x_{34} & x_{39} \\ x_{43} & x_{44} & x_{49} \\ \hline x_{73} & x_{74} & x_{79} \\ x_{83} & x_{84} & x_{89} \end{array}\right]_{4\times 3} \left[\begin{array}{cc} x_{35} & x_{36} \\ x_{45} & x_{46} \\ \hline x_{75} & x_{76} \\ x_{85} & x_{86} \end{array}\right]_{4\times 2}$$

pbdR 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 $\boldsymbol{A}$, 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 $\boldsymbol{U\Sigma V}^*$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal, and $\boldsymbol{\Sigma}$ is nonnegative and diagonal.

Stage A:

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

Stage B:

4 Form $\boldsymbol{B} = \boldsymbol{Q}^* \boldsymbol{A}$.
5 Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}} \boldsymbol{\Sigma} \boldsymbol{V}^*$.
6 Set $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.

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

Algorithm 4.4: Randomized Subspace Iteration

Given an $m \times n$ matrix $\boldsymbol{A}$ and integers ℓ and q, this algorithm computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ standard Gaussian matrix $\boldsymbol{\Omega}$.
2 Form $\boldsymbol{Y}_0 = \boldsymbol{A\Omega}$ and compute its QR factorization $\boldsymbol{Y}_0 = \boldsymbol{Q}_0 \boldsymbol{R}_0$.
3 **for** $j = 1, 2, \ldots, q$
4 Form $\widetilde{\boldsymbol{Y}}_j = \boldsymbol{A}^* \boldsymbol{Q}_{j-1}$ and compute its QR factorization $\widetilde{\boldsymbol{Y}}_j = \widetilde{\boldsymbol{Q}}_j \widetilde{\boldsymbol{R}}_j$.
5 Form $\boldsymbol{Y}_j = \boldsymbol{A}\widetilde{\boldsymbol{Q}}_j$ and compute its QR factorization $\boldsymbol{Y}_j = \boldsymbol{Q}_j \boldsymbol{R}_j$.
6 **end**
7 $\boldsymbol{Q} = \boldsymbol{Q}_q$.

Serial R

```
1  rSVD <- function(A, k, q=3)
2    {
3      ## Stage A
4      Omega <-  matrix(rnorm(n*2*k),
5          nrow=n, ncol=2*k)
6      Y <- A %*% Omega
7      Q <- qr.Q(qr(Y))
8      At <- t(A)
9      for(i in 1:q)
10       {
11         Y <- At %*% Q
12         Q <- qr.Q(qr(Y))
13         Y <- A %*% Q
14         Q <- qr.Q(qr(Y))
15       }
16
17     ## Stage B
18     B <- t(Q) %*% A
19     U <- La.svd(B)$u
20     U <- Q %*% U
21     U[, 1:k]
22   }
```

[1] Halko, Martinsson, and Tropp. 2011. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions *SIAM Review* **53** 217–288

Truncated SVD from random projections

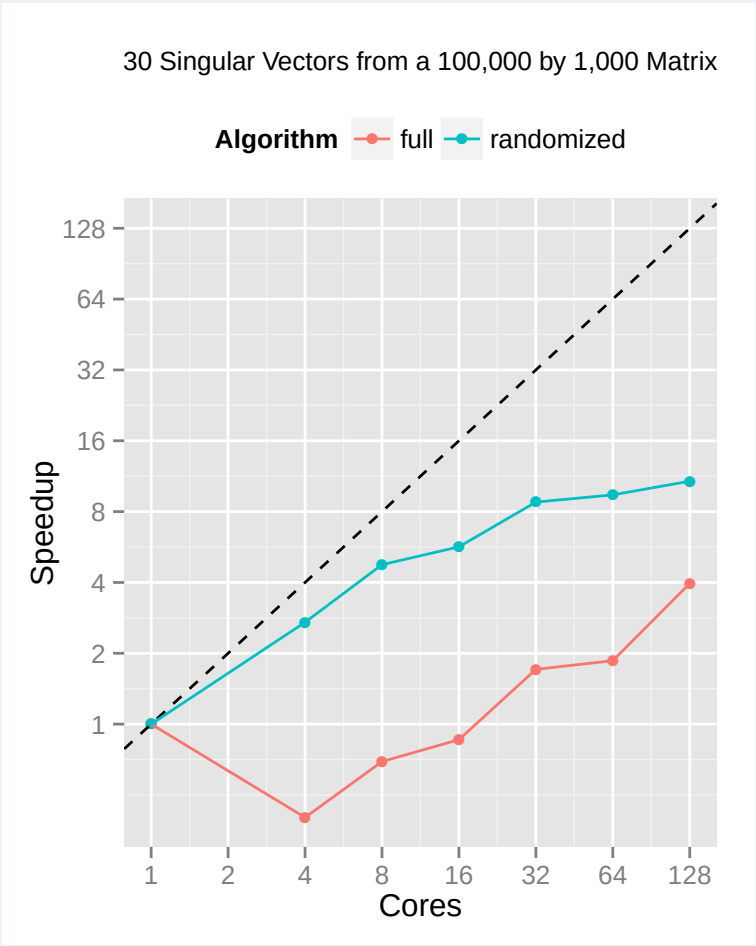
Serial R

```
1  rSVD <- function(A, k, q=3)
2    {
3      ## Stage A
4      Omega <- matrix(rnorm(n*2*k),
           nrow=n, ncol=2*k)
5      Y <- A %*% Omega
6      Q <- qr.Q(qr(Y))
7      At <- t(A)
8      for(i in 1:q)
9        {
10         Y <- At %*% Q
11         Q <- qr.Q(qr(Y))
12         Y <- A %*% Q
13         Q <- qr.Q(qr(Y))
14       }
15
16     ## Stage B
17     B <- t(Q) %*% A
18     U <- La.svd(B)$u
19     U <- Q %*% U
20     U[, 1:k]
21   }
```

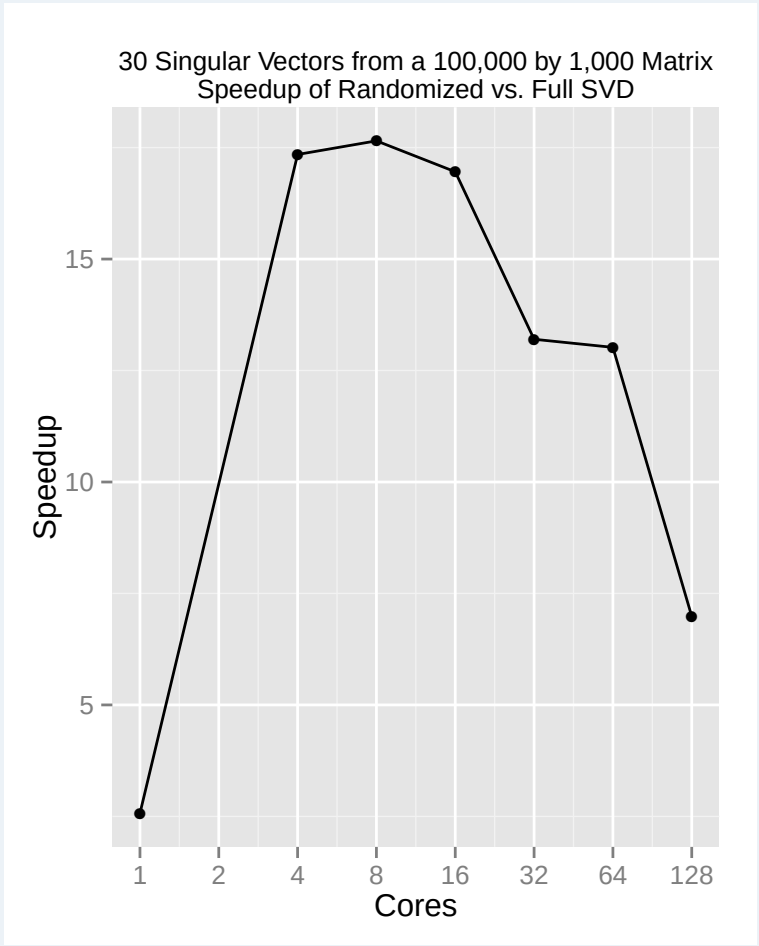
Parallel pbdR

```
1  rSVD <- function(A, k, q=3)
2    {
3      ## Stage A
4      Omega <- ddmatrix("rnorm",
5          nrow=n, ncol=2*k)
6      Y <- A %*% Omega
7      Q <- qr.Q(qr(Y))
8      At <- t(A)
9      for(i in 1:q)
10       {
11         Y <- At %*% Q
12         Q <- qr.Q(qr(Y))
13         Y <- A %*% Q
14         Q <- qr.Q(qr(Y))
15       }
16
17     ## Stage B
18     B <- t(Q) %*% A
19     U <- La.svd(B)$u
20     U <- Q %*% U
21     U[, 1:k]
22   }
```

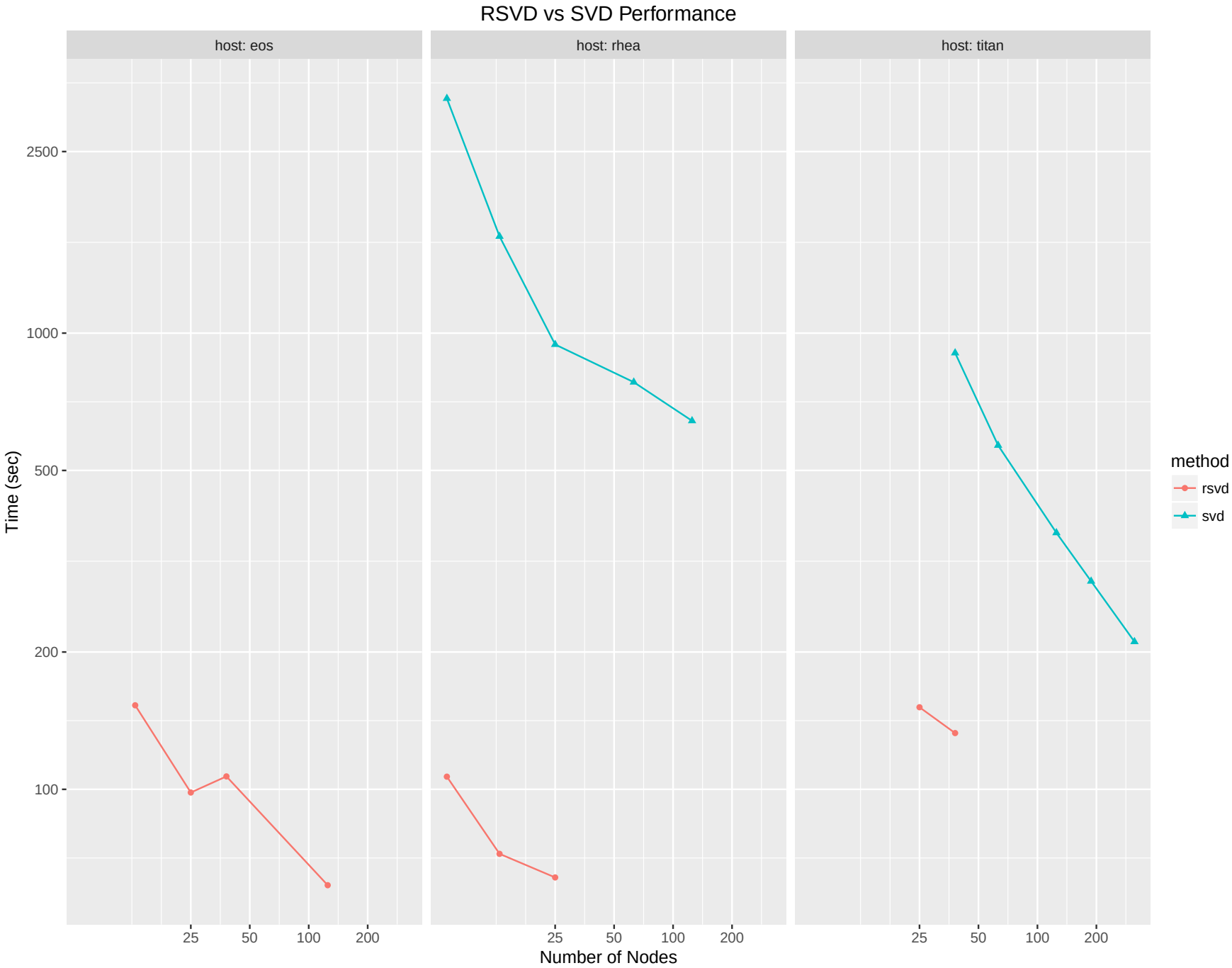
From journal to scalable code and scaling data in one day.

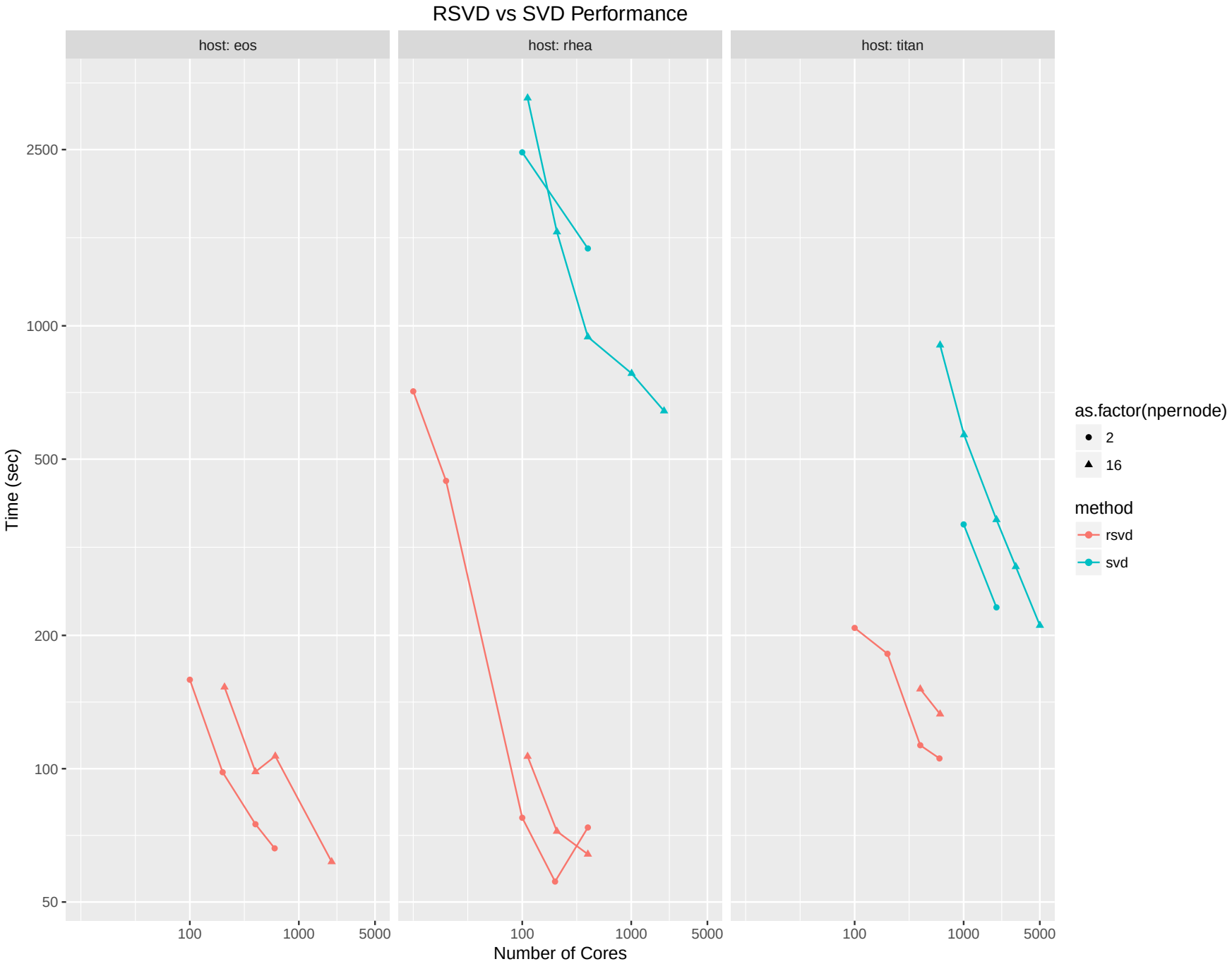


Speedup relative to 1 core



rSVD speedup relative to full SVD


RSVD vs SVD Performance
host: eos
host: rhea
host: titan
Time (sec)
2500
1000
500
200
100
25
50
100
200
Number of Nodes
method
rsvd
svd


RSVD vs SVD Performance
host: eos
host: rhea
host: titan
Time (sec)
2500
1000
500
200
100
50
100
1000
5000
Number of Cores
as.factor(npernode)
2
16
method
rsvd
svd

```
1  suppressMessages( library(rhdf5) )
2  suppressMessages( library(pbdDMAT, quiet=TRUE ) )
3  suppressMessages( library(pbdML, quiet=TRUE ) )
4
5  start.time = Sys.time( )
6  init.grid( )
7  end.time = Sys.time( )
8  barrier( )
9  comm.print( paste( "initgrid = ", end.time - start.time ) )
10
11 args = commandArgs( trailingOnly = TRUE )
12
13 meth      = args[ 1 ]
14 npernode  = strtoi( args[ 2 ] )
15 n_keep    = strtoi( args[ 3 ] )
16 block_row = strtoi( args[ 4 ] )
17 block_col = strtoi( args[ 5 ] )
18
19 nproc <- comm.size( )
20
21 rows <- 12390000 %/% nproc
22 cols <- 1250
23
24 len <- rows*cols*4
25 start <- comm.rank()*len
26
27 ## this one has individual files
28
29 if ( nproc ==     2 ) fn <- paste("X/X2", comm.rank(), "h5", sep=".")
30 if ( nproc ==     4 ) fn <- paste("X/X4", comm.rank(), "h5", sep=".")
31 if ( nproc ==     5 ) fn <- paste("X/X5", comm.rank(), "h5", sep=".")
32 if ( nproc ==    10 ) fn <- paste("X/X10", comm.rank(), "h5", sep=".")
33 if ( nproc ==    20 ) fn <- paste("X/X20", comm.rank(), "h5", sep=".")
34 if ( nproc ==   100 ) fn <- paste("X/X100", comm.rank(), "h5", sep=".")
35 if ( nproc ==   200 ) fn <- paste("X/X200", comm.rank(), "h5", sep=".")
```

```
36 if ( nproc ==   400 ) fn <- paste("X/X400", comm.rank(), "h5", sep=".")
37 if ( nproc ==   600 ) fn <- paste("X/X600", comm.rank(), "h5", sep=".")
38 if ( nproc ==  1000 ) fn <- paste("X/X1000", comm.rank(), "h5", sep=".")
39 if ( nproc ==  2000 ) fn <- paste("X3/X2000", comm.rank(), "h5", sep=".")
40 if ( nproc ==  3000 ) fn <- paste("X3/X3000", comm.rank(), "h5", sep=".")
41 if ( nproc ==  5000 ) fn <- paste("X5/X5000", comm.rank(), "h5", sep=".")
42 if ( nproc == 30000 ) fn <- paste("X/X30000", comm.rank(), "h5", sep=".")
43
44 start.time = Sys.time( )
45 A <- h5read(fn, "/dataset" )
46 end.time = Sys.time( )
47 barrier( )
48 comm.print( paste( "io = ", end.time - start.time ) )
49
50 ## comm.print( A[ 1:5, 1:5 ], all.rank = TRUE )
51 ## comm.print( dim( A ), all.rank = TRUE )
52 start.time = Sys.time( )
53 A <- new( "ddmatrix", Data=A, dim=c(12390000, 1250), ldim=dim(A), bldim=dim(A),
       ICTXT=2 )
54
55 ## comm.print( dim(submatrix( A )), all.rank = TRUE )
56 ## comm.print( submatrix( A )[ 1:5, 1:5 ], all.rank = TRUE )
57 ## comm.print( A, all.rank = TRUE )
58
59 A <- as.blockcyclic( A, bldim = c( block_row, block_col ) )
60
61 ## comm.print( A[ 1:5, 1:5 ], all.rank = TRUE )
62 ## comm.print( dim( A ), all.rank = TRUE )
63
64 end.time = Sys.time( )
65 barrier( )
66 comm.print( paste( "blockcyclic = ", end.time - start.time ) )
67
68 ## comm.print( A, all.rank = TRUE )
69 ## comm.print( submatrix( A )[ 1:5, 1:5 ], all.rank = TRUE )
```

```
70   comm.print( "Starting computation" )
71  start.time = Sys.time( )
72  if ( meth == "rsvd" )
73    Res <- rsvd( A, k = n_keep, q = 3, retu = TRUE, retv = TRUE )
74  if ( meth == "gpu_rsvd" )
75    Res <- rsvd( A, k = n_keep, q = 3, retu = TRUE, retv = TRUE )
76  if ( meth == "gpu_svd" )
77    Res <- svd( A, nu = n_keep, nv = n_keep )
78  if ( meth == "svd" )
79    Res <- svd( A, nu = n_keep, nv = n_keep )
80  end.time = Sys.time( )
81  barrier( )
82  comm.print( paste( "compute = ", end.time - start.time ) )
83
84  comm.print( Res$d )
85  mesg <- paste( "Finished ... method =", meth, "nproc =", nproc, "npernode =",
        npernode, "keep =", n_keep, "blocking =", block_row )
86  comm.print( mesg )
87
88  finalize( )
```

Where to learn more?

- http://r-pbd.org/
- **pbdDEMO** vignette
- Googlegroup:RBigDataProgramming
- 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.