Programming with Big Data in R

George Ostrouchov
Oak Ridge National Laboratory and University of Tennessee

Future Trends in Nuclear Physics Computing
March 16-18, 2016
Thomas Jefferson National Accelerator Facility
Newport News, VA
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.
### IEEE Spectrum’s 2014 Ranking of Programming Languages

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>2015 Spectrum Ranking</th>
<th>IEEE Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Java</td>
<td>![Icons]</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>2. C</td>
<td>![Icons]</td>
<td>99.9</td>
<td>99.3</td>
</tr>
<tr>
<td>3. C++</td>
<td>![Icons]</td>
<td>99.4</td>
<td>95.5</td>
</tr>
<tr>
<td>4. Python</td>
<td>![Icons]</td>
<td>96.5</td>
<td>93.5</td>
</tr>
<tr>
<td>5. C#</td>
<td>![Icons]</td>
<td>91.3</td>
<td>92.4</td>
</tr>
<tr>
<td>6. R</td>
<td>![Icons]</td>
<td>84.8</td>
<td>84.8</td>
</tr>
<tr>
<td>7. PHP</td>
<td>![Icons]</td>
<td>84.5</td>
<td>84.5</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td>![Icons]</td>
<td>83.0</td>
<td>78.9</td>
</tr>
<tr>
<td>9. Ruby</td>
<td>![Icons]</td>
<td>76.2</td>
<td>74.3</td>
</tr>
<tr>
<td>10. Matlab</td>
<td>![Icons]</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)
An Example: 

Data: 1,653 start and end timestamps for GPU offload periods.

Want 1-ahead prediction of start and end to run other codes.

```r
library(dplyr)

### start end idle busy
## 1 56114210457 56114211289 832 NA
## 2 56114300920 56114311544 10624 89631
## 3 56114373143 56114373943 800 61599
## 4 56117433146 56117436858 3712 3059203
## 5 56117469818 56117470650 832 32960
## 6 56117507098 56117517081 9983 36448
```
library(ggplot2)
ggplot(ts, aes(idle)) + geom_histogram()
ggplot(ts, aes(idle)) + geom_histogram() + scale_x_log10()
ggplot(ts, aes(busy)) + geom_histogram() + theme_bw()
ggplot(ts, aes(busy)) + geom_histogram() + scale_x_log10() + theme_bw()
ggplot(ts, aes(busy, idle, color=start)) + geom_point() + scale_x_log10() + scale_y_log10()
Successive busy idle periods cluster around several values. Markov chain ... probably sparse ... Write my own?

Turns out that most of this already exists in the package rEMM (One of 7,000+ CRAN packages!)
ts_log <- transmute(ts, lidle = log10(idle), lbusy = log10(busy))
library(rEMM)
emm <- build(EMM(threshold = 0.2, measure="euclidean"), ts_log)
plot(emm)

Markov transition graph after seeing all the data.
Add Markov node locations on top of the log scale *busy-idle* space:

```r
ggplot(ts, aes(busy, idle, color=start)) + geom_path() + geom_point() + scale_x_log10() + scale_y_log10() + geom_point(aes(10^lbusy, 10^lidle), data.frame(cluster_centers(emm)), col="red")
```

Looks promising!
pr_idle <- function(ts, threshold, lambda, measure = "euclidean")
{
    emm <- build(EMM(threshold, measure, lambda = lambda), ts_log[1:5, ])
    ts$idle_p <- ts$busy_p <- NA
    for(i in 6:nrow(ts_log))
    {
        pred_data <- cluster_centers(emm)[predict(emm, n=1), ]
        ts$idle_p[i] <- 10^(pred_data["lidle"])
        ts$busy_p[i] <- 10^(pred_data["lbusy"])
        emm <- build(emm, ts_log[i, ])
    }
    ts
}

pr_idle <- function(ts, threshold=0.1, lambda=0.01)
{
    pr_idle(ts)
}

Prediction proceeds one pair of busy, idle at a time, predicting the next pair, then updating the Markov states and probabilities for actual values observed.
library(reshape2)
ts_melt <- melt(ts, value.name="observed", measure.vars=c("idle", "busy"))
ts_melt <- mutate(ts_melt, predicted=ifelse(variable == "idle", idle_p, busy_p))
ggplot(ts_melt, aes(start, predicted/observed)) + geom_point() + scale_y_log10() +
stat_quantile(quantiles=c(.75, .25), col="blue") +
stat_quantile(quantiles=c(.95, .05), col="red") + facet_grid(~variable)

This is a lightweight algorithm in R (~2 seconds total for all predictions). It can be made more lightweight (100x ?) by implementing in C or C++ and by updating less often.
## Resources for Learning R

- **RStudio IDE**
  

- **Task Views**: [http://cran.at.r-project.org/web/views](http://cran.at.r-project.org/web/views)

- **Book**: *The Art of R Programming* by Norm Matloff:  
  [http://nostarch.com/artofr.htm](http://nostarch.com/artofr.htm)

- **Advanced R**: [http://adv-r.had.co.nz/](http://adv-r.had.co.nz/) and *ggplot2*  
  [http://docs.ggplot2.org/current/](http://docs.ggplot2.org/current/) by Hadley Wickham

- **R programming for those coming from other languages**:  

- **aRrgh: a newcomer’s (angry) guide to R**, by Tim Smith and Kevin Ushey:  
  [http://tim-smith.us/arrgh/](http://tim-smith.us/arrgh/)


- **The [R] stackoverflow tag.**
Why R?: Programming with Big Data

Wei-Chen Chen\textsuperscript{1}
George Ostrouchov\textsuperscript{2,3,4}
Pragneshkumar Patel\textsuperscript{3}
Drew Schmidt\textsuperscript{4}

\textsuperscript{1}FDA
Washington, DC, USA

\textsuperscript{2}Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge TN, USA

\textsuperscript{3}Joint Institute for Computational Sciences
University of Tennessee, Knoxville TN, USA

\textsuperscript{4}Business Analytics and Statistics
University of Tennessee, Knoxville TN, USA
HPC Cluster with NVRAM and Parallel File System

Today’s HPC Cluster

Parallel File System

Compute Nodes

I/O Nodes

Storage Servers

Disk

Login Nodes

Your Laptop

Big Data

“Little Data”
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>
SPMD: Copies of One Code Run Asynchronously

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

Example: Random Forest Code
(build many simple models, 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_rf_p.r

```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")
```
Dense Matrix and Vector Operations

A matrix is mapped to a processor grid shape

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

- (a) \(1 \times 6\)
- (b) \(2 \times 3\)
- (c) \(3 \times 2\)
- (d) \(6 \times 1\)

**Table:** Processor Grid Shapes with 6 Processors
The above (and over 100 other functions) runs on 1 core with R or 10,000 cores with pbdR ddmatrix class

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

```r
> showClass("ddmatrix")
Class "ddmatrix" [package "pbdDMAT"]
Slots:
Name:  Data    dim    ldim    bldim    ICTXT
Class:  matrix numeric numeric numeric numeric numeric
```
Truncated SVD from random projections

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-2\( k \) 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 \( Y = (AA^*)^qA\Omega \) 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: \( B = \tilde{U} \Sigma V^* \).
6. Set \( U = \tilde{Q} \).

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 \( Y_0 = Q_0R_0 \).
3. For \( j = 1, 2, \ldots, q \):
   1. Form \( \bar{Y}_j = A^*Q_{j-1} \) and compute its QR factorization \( \bar{Y}_j = \bar{Q}_j\bar{R}_j \).
   2. Form \( Y_j = A\bar{Q}_j \) and compute its QR factorization \( Y_j = Q_jR_j \).
4. \( Q = Q_q \).

Serial R

```r
randSVD <- 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
randSVD <- 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]
}
```

Parallel pbdR

```r
randSVD <- 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))
        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]
}
```
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

**Graph 1:**
- Speedup relative to 1 core
- Cores: 1, 2, 4, 8, 16, 32, 64, 128
- Speedup: 1, 2, 4, 8, 16, 32, 64, 128

**Graph 2:**
- RandSVD speedup relative to full SVD
- Cores: 1, 2, 4, 8, 16, 32, 64, 128
- Speedup: 1, 2, 4, 8, 16, 32, 64, 128

---

Speedup of Randomized vs. Full SVD

Speedup relative to 1 core

RandSVD speedup relative to full SVD
Future Work

- NSF/DMS: second year of a 3 year grant to
  - Bring back interactivity via client/server (pbdCS/pbdZMQ)
  - Simplify parallel data input
  - Begin DPLASMA integration
  - Outreach to the statistics community
- DOE/SC: In-situ or staging use with simulations
  - pbdADIOS - HPC I/O
- Pending: NSF BIGDATA, Tensor Regression
- Pending: Exascale Computing Project, analytics for ParaView/VisIt
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
- Need access to a cluster computer? From NSF:
  - XSEDE *trial* or *startup* allocation
    - [https://www.xsede.org/web/xup/allocations-overview](https://www.xsede.org/web/xup/allocations-overview)
  - Most resources have `pbdR` installed

Support

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

This work used resources of the [Oak Ridge Leadership Computing Facility](https://www.ornl.gov) 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 also used resources of [National Institute for Computational Sciences](https://www.nics.tennessee.edu) at the University of Tennessee, Knoxville, which is supported by the U.S. National Science Foundation.