pbdR: Harnessing HPC Research for Parallel Computing with R

George Ostrouchov
Oak Ridge National Laboratory and University of Tennessee

ISM HPCCON and ISM HPC on R Workshop, October 9-12, 2015
Institute of Statistical Mathematics, Tokyo, Japan
Wei-Chen Chen¹
George Ostrouchov²,³
Pragneshkumar Patel³
Drew Schmidt³

¹FDA
Washington, DC, USA

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

³Joint Institute for Computational Sciences
University of Tennessee, Knoxville TN, USA

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 National Institute for Computational Sciences at the University of Tennessee, Knoxville, which is supported by the Office of Cyberinfrastructure of the U.S. National Science Foundation. This work also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
1 Introduction to HPC and Its View from R
   - Three Basic Flavors of Parallel Hardware
   - Cluster Computer Architectures
   - A Quick Overview of Parallel Software
   - Batch and Interactive
   - Programming Models

2 pbdR
   - The pbdR Project
   - pbdMPI
   - pbdDMAT
   - RandSVD
   - pbdMPI Example: Random Forest Prediction
   - pbdMPI Example: Functional Data Analysis

3 pbdCS
   - Client-Server Demo

4 Future Work
Contents

1 Introduction to HPC and Its View from R
   - Three Basic Flavors of Parallel Hardware
   - Cluster Computer Architectures
   - A Quick Overview of Parallel Software
   - Batch and Interactive
   - Programming Models
1 Introduction to HPC and Its View from R

- Three Basic Flavors of Parallel Hardware
  - Cluster Computer Architectures
  - A Quick Overview of Parallel Software
  - Batch and Interactive
  - Programming Models
Cores and Co-Processors to Nodes

Distributed Memory

Interconnection Network

PROC + cache
PROC + cache
PROC + cache
PROC + cache
Mem
Mem
Mem
Mem

Shared Memory

CORE + cache
CORE + cache
CORE + cache
CORE + cache
Network
Memory

Co-Processor

Local Memory

GPU: Graphical Processing Unit
MIC: Many Integrated Core
Introduction to HPC and Its View from R

- Three Basic Flavors of Parallel Hardware
- Cluster Computer Architectures
- A Quick Overview of Parallel Software
- Batch and Interactive
- Programming Models
Parallel Computing before Multicore

HPC “Beowulf” Clusters before 2005

Software Developments:
- MPI is mature, MapReduce emerges
- Parallel Libraries: PBLAS, ScaLAPACK, PETSc, etc.
- Resource Manager: PBS mature, HADOOP emerges

HPC “Beowulf” Clusters before 2005
Compute Nodes and Disk
Login Nodes
Your Laptop
"Little Data"

Big Data
Multicore Emerges and Clusters become Diskless

2005-2015 HPC Cluster

Parallel File System

Disk Storage

Servers

Compute Nodes

I/O Nodes

Login Nodes

Your Laptop

Big Data

“Little Data”

Software Developments

OpenMP, CUDA, OpenCL, OpenACC

Libraries: PLASMA, MAGMA, CuBLAS
Adding NVRAM to New HPC Systems

Today's HPC Cluster

Parallel File System

Solid State Disk

Big Data

Software Developments

Libraries: DPLASMA, CombBLAS
HADOOP fades, Spark emerges

Compute Nodes

I/O Nodes

Storage Servers

Disk

Login Nodes

Your Laptop

“Little Data”
Introduction to HPC and Its View from R

- Three Basic Flavors of Parallel Hardware
- Cluster Computer Architectures
- A Quick Overview of Parallel Software
- Batch and Interactive
- Programming Models
“Native” Programming Models and Tools

**Distributed Memory**

- Default is parallel (SPMD): what is my data and what do I need from others?
- Offload data and tasks. We are slow but many!

**Shared Memory**

- Default is serial: which tasks can the compiler make parallel?
- Co-Processor

**Interconnection Network**

**PROC + cache**

**Mem**

**CORE + cache**

**Network**

**Memory**

**Local Memory**

**CUDA**

**OpenCL**

**OpenACC**

**OpenMP**

**Pthreads**

**fork**

**Sockets**

**MPI**

**MapReduce**
Distributed Programming Works in Shared Memory

- Default is parallel (SPMD): what is my data and what do I need from others?
- Default is serial: which tasks can the compiler make parallel?

Co-Processor
- GPU: Graphical Processing Unit
- MIC: Many Integrated Core

Offload data and tasks.
- We are slow but many!

Local Memory
- Offload data and tasks.

Interconnection Network
- PROC + cache
- Mem

Distributed Memory
- PROC + cache
- PROC + cache
- PROC + cache
- PROC + cache

Network
- CORE + cache
- CORE + cache
- CORE + cache
- CORE + cache

Memory
- CORE + cache
- CORE + cache
- CORE + cache
- CORE + cache

Sockets
- MPI
- MapReduce

CUDA
- OpenCL
- OpenACC
- OpenMP
- Pthreads
- fork

✔
- Local Memory

✘
- Co-Processor

✘
- GPU: Graphical Processing Unit
- MIC: Many Integrated Core

Harnessing HPC Research for R
R Interfaces to Low-Level Native Tools

- **Distributed Memory**
  - Interconnection Network
  - Proc + cache
  - Mem

- **Shared Memory**
  - CORE + cache
  - Network
  - Memory

**Default is parallel (SPMD): what is my data and what do I need from others?**

**Default is serial: which tasks can the compiler make parallel?**

- Sockets
- MPI
- MapReduce
- R/Hadoop, SparkR
- Snow
- Rmpi
- Rhpc
- PbdMPI
- CUDA
- OpenCL
- OpenACC
- OpenMP
- Pthreads
- Fork
- Multicore

**Offload data and tasks. We are slow but many!**

- Foreign Language Interfaces: `.C`, `.Call`, Rcpp, OpenCL, inline

**snow + multicore = parallel**
Introduction to HPC and Its View from R

A Quick Overview of Parallel Software

R and \textbf{pbdR} Interfaces to HPC Libraries

- **Local Memory**
- **Co-Processor**
  - GPU: Graphical Processing Unit
  - MIC: Many Integrated Core
- **Interconnection Network**

**Shared Memory**
- ACML (AMD)
- LibSci (Cray)
  - MKL (Intel)

**Distributed Memory**
- ScaLAPACK
- BLAS
- PLASMA
- DPLASMA
- MAGMA
- CombBLAS

**Profiling**
- Tau
- mpiP
- fpmpi
- PAPI

**I/O**
- NetCDF4
- ADIOS

**Learning**
- pbdDEMO

- **ZeroMQ**
- **MPI**
- **PETSc**
- **Trilinos**
- **PAPI**

- **Distributed Memory**
  - \textbf{pbdMPI}

- **Shared Memory**
  - \textbf{ACML} (AMD)

- **R**

- Released
- Under Development

- \textbf{pbdR Core Team}

Harnessing HPC Research for R

8/43
1 Introduction to HPC and Its View from R
- Three Basic Flavors of Parallel Hardware
- Cluster Computer Architectures
- A Quick Overview of Parallel Software
- Batch and Interactive
- Programming Models
Data analysis is interactive!

- Data reduction to knowledge
- Iterative process with same data
  - Exploration, model construction
  - Diagnostics of fit and quantification of uncertainty
  - Interpretation
- S (and R) interactive “answer” to batch data analysis
- Efficient use of expensive people

Big platform computing is batch!

- Libraries built for batch computing
- Traditionally data generation by simulation science
- Efficient use of expensive platforms
High-Level Language: Batch and Interactive Distinction Blurred.

- A function is a “batch” script
- R “An interactive environment to use batch scripts”

Ideal solution: Interactive Client with a Batch Server

- Parallel visualization systems (VisIt and ParaView) are client-server (batch on server)
- Current pbdR packages address server side (batch)
- pbdCS 0.1-0 released on GitHub
  - Interactive SPMD
  - Based on ZeroMQ distributed messaging (pbdZMQ 0.1-1 on CRAN)
  - Bridge resource manager (pbdSCHED 0.1-0 on GitHub)
  - Site configuration file
  - Manage relationship of big data (server side) to little data (client side)
Introduction to HPC and Its View from R

- Three Basic Flavors of Parallel Hardware
- Cluster Computer Architectures
- A Quick Overview of Parallel Software
- Batch and Interactive
- Programming Models
Manager-Workers

- A serial program (Manager) divides up work and/or data
- Workers run in parallel without interaction
- Manager collects/combines results from workers
- Divide-Recombine fits this model
MapReduce

- A concept born of a search engine
- Decouples certain coupled problems with an intermediate communication - shuffle
- User writes two serial codes: Map and Reduce
MapReduce: a Parallel Search Engine Concept

<table>
<thead>
<tr>
<th>Search MANY documents</th>
<th>Serve MANY users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Pages (records)</td>
<td>Web Pages (records)</td>
</tr>
<tr>
<td>p0</td>
<td>p0</td>
</tr>
<tr>
<td>p1</td>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
<td>p2</td>
</tr>
<tr>
<td>p3</td>
<td>p3</td>
</tr>
</tbody>
</table>

Index Words (keys)

\[
\begin{bmatrix}
A_1 & A_2 & A_3 & A_4 \\
B_1 & B_2 & B_3 & B_4 \\
C_1 & C_2 & C_3 & C_4 \\
D_1 & D_2 & D_3 & D_4 \\
\end{bmatrix}
\]

Shuffle

\[
\text{MPI Alltoallv}
\]

Index Words (keys)

\[
\begin{bmatrix}
A_1 & B_1 & C_1 & D_1 \\
A_2 & B_2 & C_2 & D_2 \\
A_3 & B_3 & C_3 & D_3 \\
A_4 & B_4 & C_4 & D_4 \\
\end{bmatrix}
\]

Matrix transpose in another language?
Can use different sets of processors

<table>
<thead>
<tr>
<th>Web Pages (records)</th>
<th>Index Words (keys)</th>
<th>Streaming Shuffle</th>
<th>Web Pages (records)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>B1 B2 B3 B4</td>
<td>→</td>
<td>B1</td>
</tr>
<tr>
<td>p1</td>
<td></td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td>p2</td>
<td></td>
<td>MPI_Scatter</td>
<td>B3</td>
</tr>
<tr>
<td>p3</td>
<td></td>
<td></td>
<td>B4</td>
</tr>
</tbody>
</table>
MPI and MapReduce

Both Concepts are about Communication

- One makes communication explicit, gives choices
- The other hides communication, gives one choice (shuffle)
SPMD: Single Program Multiple Data

- The prevalent way of distributed programming
- Can handle tightly coupled parallel computations
- It is designed for batch computing
- There is usually no manager - rather, all cooperate
- Prime driver behind MPI specification
Early SPMD Work in Statistics: Crossproduct (Row-Block)

Fig. 4. Computation of $A = X'X$ on an 8-processor hypercube, with final result on processor 0.

Fig. 6. Computation of $A = X'X$ on an 8-processor hypercube, with final result on all processors.

Hypercube: Individual send() and recv() over each dimension

Simplified with MPI (and further with pbdMPI)

Fig. 4. Computation of $A = XX$ on an 8-processor hypercube, with final result on processor 0.

Fig. 6. Computation of $A = XX$ on an 8-processor hypercube, with final result on all processors.

Architecture-specific vendor optimizations

- Cray MPT
- SGI MPT
Data-flow: Parallel Runtime Scheduling and Execution Controller (PaRSEC)

- Master data-flow controller runs distributed on all cores.
- Dynamic generation of current level in flow graph
- Effectively removes collective synchronizations

Contents

2 pbdR
- The pbdR Project
- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
The pbdR Project

- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
Why use HPC libraries?

- The libraries represent 30+ years of research by the HPC community
- They’re tested. They’re fast. They’re scalable.
- Many science communities are invested in their API.
- HPC Simulation Science uses much of the same math as data analysis
2 pbdR

- The pbdR Project
- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
pbdMPI: Simplified, Extensible, and Fast Communication Operations

- S4 methods for collective communication: extensible to other R objects.
- Default methods (like `Robj` in `Rmpi`) check for data type: safe for general users.
- API is simplified: defaults in control objects.
- Array and matrix methods without serialization: faster than `Rmpi`.

<table>
<thead>
<tr>
<th>pbdMPI (S4)</th>
<th>Rmpi</th>
</tr>
</thead>
<tbody>
<tr>
<td>allgather</td>
<td>mpi.allgather, mpi.allgatherv, mpi.allgather.Robj</td>
</tr>
<tr>
<td>allreduce</td>
<td>mpi.allreduce</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>

Harnessing HPC Research for R 21/43
**Integer?** Not always obvious in R.

```r
> is.integer(1)
[1] FALSE
> is.integer(2)
[1] FALSE
> is.integer(1:2)
[1] TRUE
```

**pbdMPI lets R figure it out**

```r
# int
mpi.allreduce(x, type=1)
# double
mpi.allreduce(x, type=2)
```

```r
allreduce(x)
```
Single Program (SPMD): Runs Asynchronous Parallel

Rank Query Example

```r
1_rank.r

library(pbdMPI, quiet = TRUE)
init()

my.rank <- comm.rank()
comm.print(my.rank, all.rank=TRUE)

finalize()
```

Execute this batch script via:
```
mpirun -np 2 Rscript 1_rank.r
```

Sample Output:
```
COMM.RANK = 0
[1] 0
COMM.RANK = 1
[1] 1
```
pbdR

- The pbdR Project
- pbdMPI
- **pbdDMAT**
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
Mapping a Matrix to Processors

### Processor Grid Shapes

<table>
<thead>
<tr>
<th>Shape</th>
<th>Description</th>
<th>Matrix</th>
</tr>
</thead>
</table>
| (a) 1×6 | 1×6 grid | \[
\begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 \\
\end{bmatrix}
\] |
| (b) 2×3 | 2×3 grid | \[
\begin{bmatrix}
0 & 1 & 2 \\
3 & 4 & 5 \\
\end{bmatrix}
\] |
| (c) 3×2 | 3×2 grid | \[
\begin{bmatrix}
0 & 1 \\
2 & 3 \\
4 & 5 \\
\end{bmatrix}
\] |
| (d) 6×1 | 6×1 grid | \[
\begin{bmatrix}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{bmatrix}
\] |

**Table:** Processor Grid Shapes with 6 Processors
2×3 block-cyclic grid on 6 processors: Global view “ddmatrix” class

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

Processor grid = \[
\begin{bmatrix}
0 & 1 & 2 \\
3 & 4 & 5
\end{bmatrix} = \begin{bmatrix}
(0,0) & (0,1) & (0,2) \\
(1,0) & (1,1) & (1,2)
\end{bmatrix}
\]
2×3 block-cyclic grid on 6 processors: Local view “ddmatrix” class

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

Processor grid = \[
\begin{bmatrix}
0 & 1 & 2 \\
3 & 4 & 5
\end{bmatrix}
= \begin{bmatrix}
(0,0) & (0,1) & (0,2) \\
(1,0) & (1,1) & (1,2)
\end{bmatrix}
\]
**Example Syntax**

```
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 numeric
```

```
> x <- as.rowblock(x)
> x <- as.colblock(x)
> x <- redistribute(x, bldim=c(8, 8), ICTXT = 0)
```
pbdDMAT Scalability Benchmarks

- Default choices throughout (no MKL, ACML, etc.)
- 1 core = 1 MPI process (Kraken: 6-core Opterons)
- Generate random matrix
  - Global Columns: 500, 1000, and 2000
  - Global Rows: fixed per core to make 43.4MiB
- Measure wall clock time
- “weak scaling” = global problem grows with core count
**pbdDMAT Scalability Benchmarks**

```r
x <- ddmatrix("rnorm", nrow=n, ncol=p)
cov.x <- cov(x)
```

```r
b <- ddmatrix("runif", nrow=p, ncol=1)
y <- x %*% b
b.hat <- lm.fit(x, y)$coefficients
```

---

**Graphs**

- **First Graph:** Comparison of run times for different predictor counts and core counts. The graph shows the scalability of the `ddmatrix` function for varying core counts and predictor counts.

- **Second Graph:** Comparison of run times for different predictor counts and core counts. The graph shows the scalability of the `lm.fit` function for varying core counts and predictor counts.
Matrix Exponentiation (pbdDMAT)

- Fitting biogeography models requires many matrix exponentiations
- Benchmark: Matrix exponential of 5000×5000 matrix.
- R 3.1.0, Matrix 1.1-2, rexpokit 0.25, pbdDMAT 0.3-0
- Libs: Cray LibSci, NETLIB ScaLAPACK, Compilers: gnu 4.8.2
- Configuration: 1 thread == 1 MPI rank == 1 physical core

2 pbdR
- The pbdR Project
- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
Randomized truncated SVD

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 \( Y = (AA^*)^qA\Omega \) 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 \( U = Q\tilde{U} \).

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

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]
}
```

Randomized truncated SVD

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))
  }
  ## Stage B
  B <- t(Q) %*% A
  U <- La.svd(B)$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))
  }
  ## Stage B
  B <- t(Q) %*% A
  U <- La.svd(B)$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

RandSVD speedup relative to full SVD

Speedup of Randomized vs. Full SVD
2 pbdR

- The pbdR Project
- pbdMPI
- pbdDMAT
- RandSVD

- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
**Letter Recognition Data**

Example 1: Letter Recognition data from package *mlbench* $(20,000 \times 17)$

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

Example 1: Random Forest Algorithm

1. Build simple regression trees from random subsets of columns
2. Use model averaging for prediction
3. Package **randomForest** has a `combine()` function that enables the following parallel approach:
   1. Everyone gets the same training data
   2. Split regression tree building among processors (**randomForest**)
   3. Use `allgather` to bring built predictors to all
   4. Everyone combine predictors
   5. Split prediction work by blocks of rows
   6. Use `allreduce` to assess prediction
4. Steps (3) and (4) can be improved with a custom reduce/combine to take advantage of MPI vendor optimizations
Example 1: Random Forest Code
(Split learning by blocks of trees. Split prediction by blocks of rows.)

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 1: Random Forest Code
(Split learning by blocks of trees. Split prediction by blocks of rows.)

Parallel Code 4_r_f_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,]

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")
```

```
Runs serial or on any number of cores

```r
[beacon-login2 stats]$ time Rscript 4_rf_s.r
Proportion Correct: 0.96725
real 0m49.028s  user 0m48.626s  sys 0m0.335s

[beacon-login2 stats]$ time Rscript 4_rf_p.r
Proportion Correct: 0.96425
real 0m52.634s  user 0m51.914s  sys 0m0.598s

[beacon-login2 stats]$ time mpirun -np 2 Rscript 4_rf_p.r
Proportion Correct: 0.96425
real 0m28.349s  user 0m54.570s  sys 0m1.070s

[beacon-login2 stats]$ time mpirun -np 4 Rscript 4_rf_p.r
Proportion Correct: 0.963
real 0m16.380s  user 1m19.301s  sys 0m3.421s

[beacon-login2 stats]$ time mpirun -np 8 Rscript 4_rf_p.r
Proportion Correct: 0.963
real 0m11.010s  user 1m19.301s  sys 0m3.421s

[beacon-login2 stats]$ time mpirun -np 16 Rscript 4_rf_p.r
Proportion Correct: 0.9635
real 0m10.655s  user 2m32.508s  sys 0m6.624s

[beacon-login2 stats]$ time mpirun -np 32 Rscript 4_rf_p.r
Proportion Correct: 0.96325
real 0m21.692s  user 4m44.114s  sys 0m20.179s
```
2 pbdR
- The pbdR Project
- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis
### Profiling min.basis()

```r
> summaryRprof()

```

<table>
<thead>
<tr>
<th>Function</th>
<th>Total Time</th>
<th>Total Pct</th>
<th>Self Time</th>
<th>Self Pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;min.basis&quot;</td>
<td>12.32</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>&quot;type.CV&quot;</td>
<td>6.54</td>
<td>53.08</td>
<td>0.02</td>
<td>0.16</td>
</tr>
<tr>
<td>&quot;S.basis&quot;</td>
<td>5.76</td>
<td>46.75</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>&quot;drop&quot;</td>
<td>4.20</td>
<td>34.09</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>&quot;norm.fdata&quot;</td>
<td>4.20</td>
<td>34.09</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>&quot;metric&quot;</td>
<td>4.18</td>
<td>33.93</td>
<td>1.04</td>
<td>8.44</td>
</tr>
<tr>
<td>&quot;%*%&quot;</td>
<td>3.98</td>
<td>32.31</td>
<td>3.98</td>
<td>32.31</td>
</tr>
<tr>
<td>&quot;getbasispenalty&quot;</td>
<td>2.72</td>
<td>22.08</td>
<td>0.02</td>
<td>0.16</td>
</tr>
<tr>
<td>&quot;bsplinepen&quot;</td>
<td>2.68</td>
<td>21.75</td>
<td>0.36</td>
<td>2.92</td>
</tr>
<tr>
<td>&quot;int.simpson2&quot;</td>
<td>2.54</td>
<td>20.62</td>
<td>1.96</td>
<td>15.91</td>
</tr>
<tr>
<td>&quot;t&quot;</td>
<td>2.10</td>
<td>17.05</td>
<td>0.10</td>
<td>0.81</td>
</tr>
<tr>
<td>&quot;ppBspline&quot;</td>
<td>1.60</td>
<td>12.99</td>
<td>0.82</td>
<td>6.66</td>
</tr>
</tbody>
</table>
```

...
Example: `min.basis()` 110 lines

```r
min.basis <- function(fdataobj, type.CV = GCV.S, ...,)
{
  ...
  library(pbdMPI)
  init()
  my.k <- get.jid(lenlambda)
  my.gcv <- array(Inf, dim = c(lenbasis, length(my.k)))
  ...
  for (i in 1:lenbasis) {
    ...
    for (k in my.k) {
      S2 <- S.basis(tt, base, lambda[k])
      my.gcv[i, k - my.k[1] + 1] <-
        type.CV(fdataobj, S = S2, W = W, trim =
        par.CV$trim, draw = par.CV$draw, ...)
    }
  }
  gcv <- do.call(cbind, allgather(my.gcv))
  finalize()
  ...
}
```

SPMD: Add 5, change 3
Contents

3 pbdCS
- Client-Server Demo
Client-Server Demo
Some explanation goes here The demo goes here
Future Work
Future Work

- Second year of a 3 year NSF grant to
  - Bring back interactivity via client/server (pbdCS 0.1-0)
  - Simplify parallel data input
  - Begin DPLASMA integration
  - Outreach to the statistics community
- DOE funding: In-situ or staging use with simulations
  - Machine learning from fusion simulation data
- Collaboration wishlist
  - RDD, HDFS, etc., file readers
  - Communicator integration with SparkR or Spark
  - Communicator integration with VisIt and ParaView
  - pbdCS integration with RStudio IDE
  - Instrumentation of various R packages with pbdR
Where to learn more?

- [http://r-pbd.org/](http://r-pbd.org/)
- pbdDEMO vignette
- Googlegroup:RBigDataProgramming