

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



The pbdR Core Team

Wei-Chen Chen¹
George Ostrouchov^{2,3}
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.



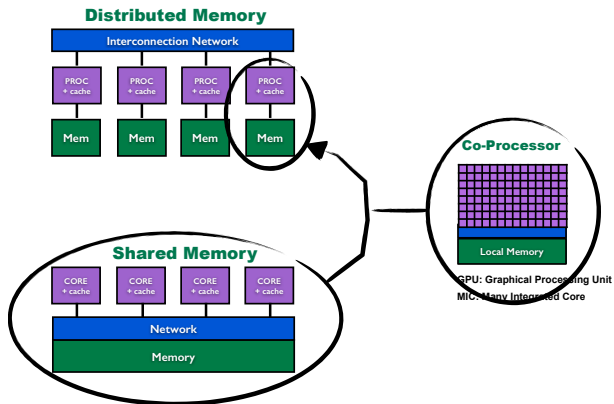
- 1 Introduction to HPC and Its View from R
 - Three Basic Flavors of Parallel Hardware
 - Cluster Computer Architectures
 - A Quick Overview of Parallel Software
 - Batch and Interactive
 - Programming Models
- 2 pbdR
 - The pbdR Project
 - pbdMPI
 - pbdDMAT
 - RandSVD
 - pbdMPI Example: Random Forest 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

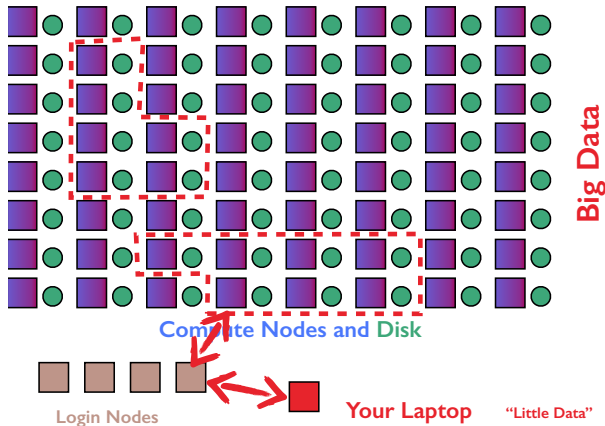
Cores and Co-Processors to Nodes



- 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

Parallel Computing before Multicore

HPC “Beowulf” Clusters before 2005



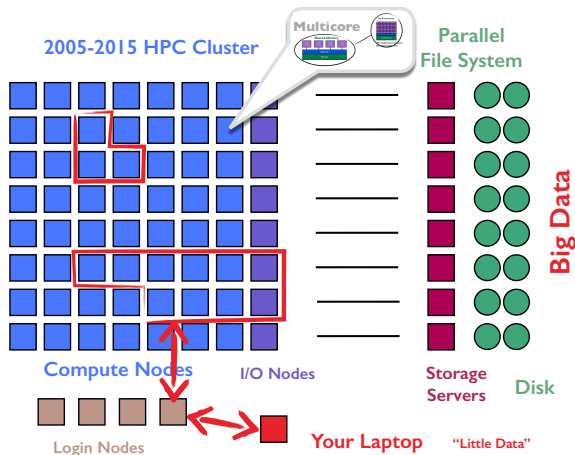
Software Developments:

MPI is mature, MapReduce emerges

Parallel Libraries: PBLAS, ScaLAPACK, PETSc, etc.

Resource Manager: PBS mature, HADOOP emerges

Multicore Emerges and Clusters become Diskless

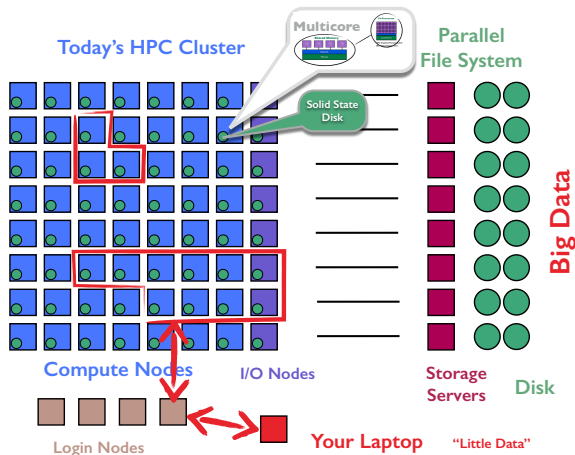


Software Developments

OpenMP, CUDA, OpenCL, OpenACC

Libraries: PLASMA, MAGMA, CuBLAS

Adding NVRAM to New HPC Systems

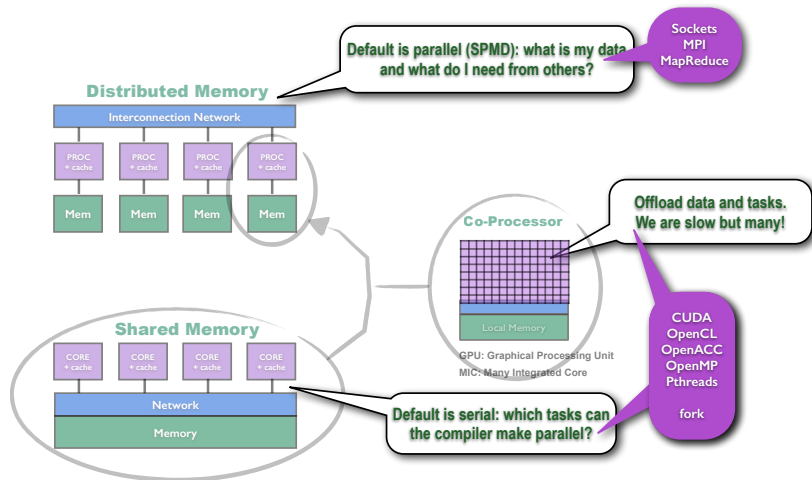


Software Developments

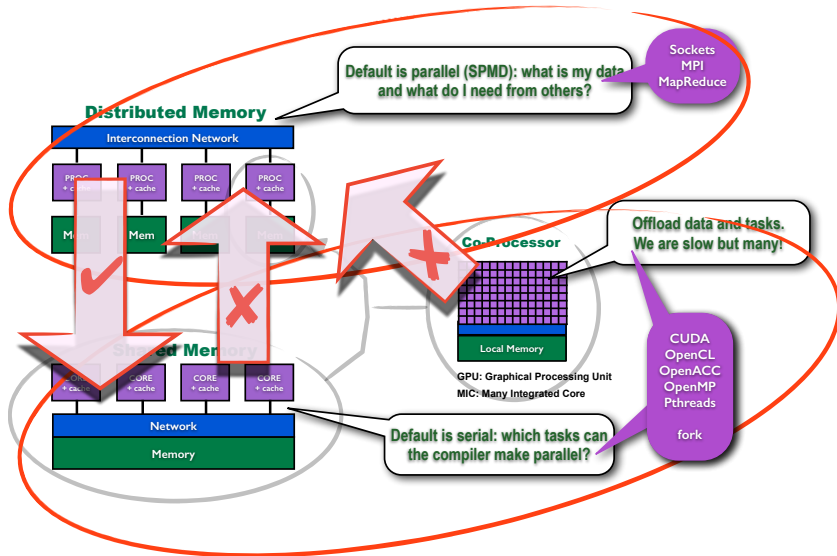
Libraries: DPLASMA,
CombBLAS
HADOOP fades, Spark
emerges

- 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

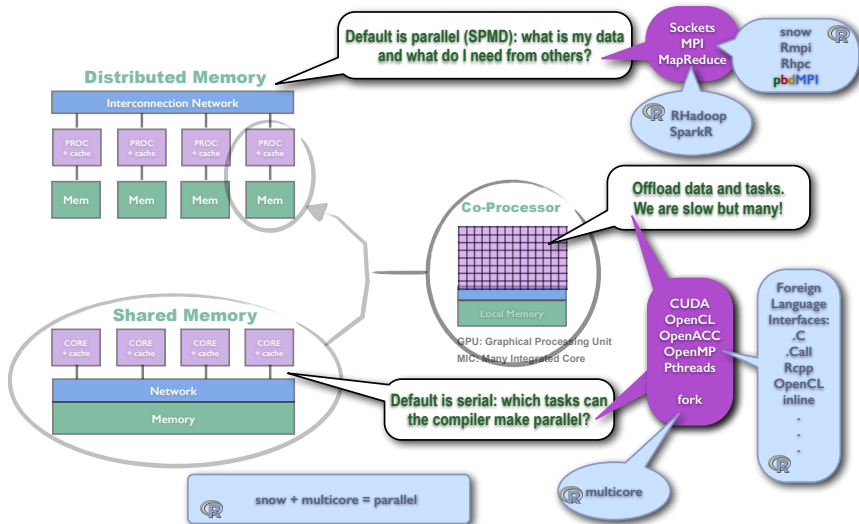
"Native" Programming Models and Tools



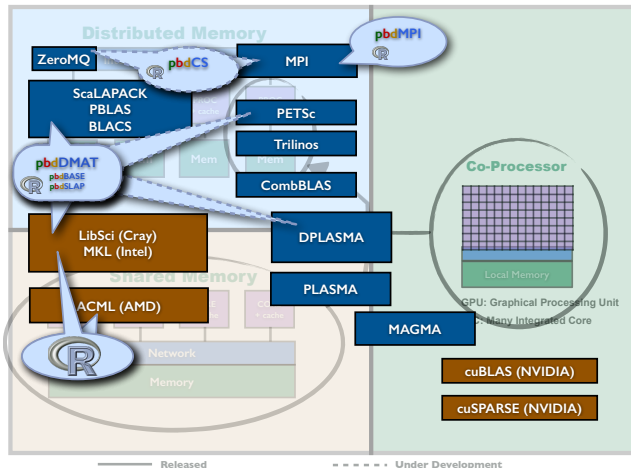
Distributed Programming Works in Shared Memory



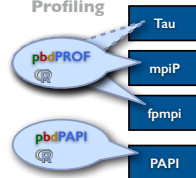
R Interfaces to Low-Level Native Tools



R and pbdR Interfaces to HPC Libraries



Profiling



I/O



Learning



- 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 **pb**dR 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)

- 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

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

Search MANY documents

Serve MANY users

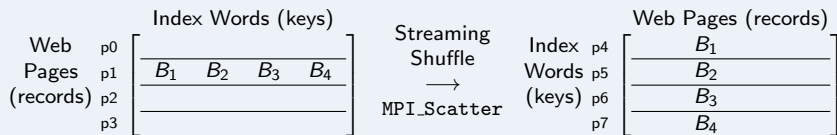
		Index Words (keys)			
Web Pages (records)	p0	A_1	A_2	A_3	A_4
	p1	B_1	B_2	B_3	B_4
	p2	C_1	C_2	C_3	C_4
	p3	D_1	D_2	D_3	D_4

Shuffle
→
MPI_Alltoallv

		Web Pages (records)			
Index Words (keys)	p0	A_1	B_1	C_1	D_1
	p1	A_2	B_2	C_2	D_2
	p2	A_3	B_3	C_3	D_3
	p3	A_4	B_4	C_4	D_4

Matrix transpose in another language?

Can use different sets of processors



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)

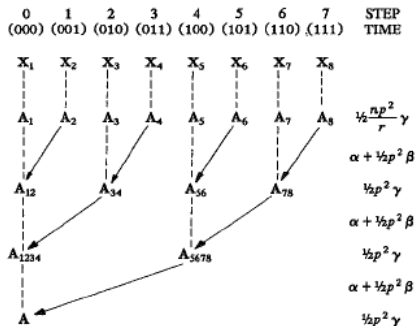


FIG. 4. Computation of $A = X'X$ on an 8-processor hypercube, with final result on processor 0.

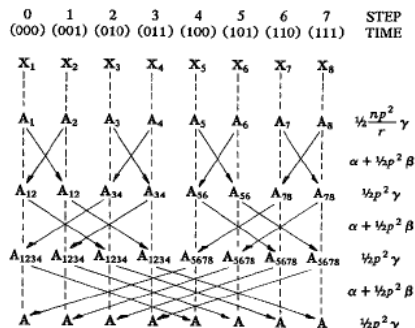


FIG. 6. Computation of $A = X'X$ on an 8-processor hypercube, with final result on all processors.

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

Ostrouchov (1987). Parallel Computing on a Hypercube: An overview of the architecture and some applications. *Proceedings of the 19th Symposium on the Interface of Computer Science and Statistics*, p.27-32.

Simplified with MPI (and further with pbdMPI)

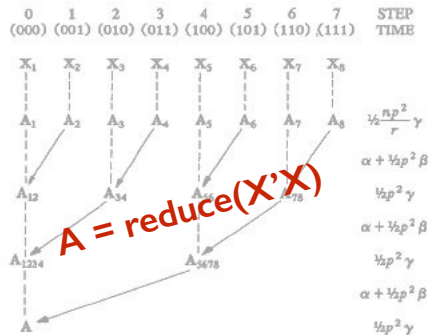


FIG. 4. Computation of $A = X'X$ on an 8-processor hypercube, with final result on processor 0.

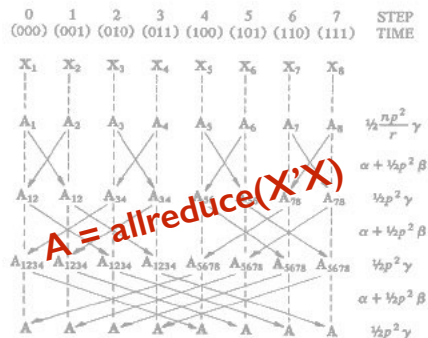


FIG. 6. Computation of $A = X'X$ on an 8-processor hypercube, with final result on all processors.

Architecture-specific vendor optimizations

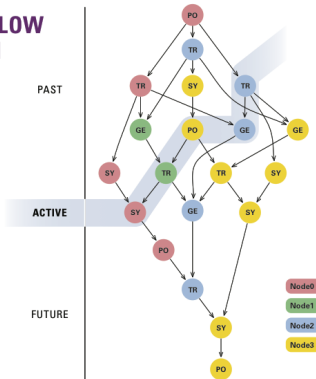
- Cray MPT
- SGI MPT

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

EFFICIENT DATA FLOW REPRESENTATION

FEATURES

- Supports Distributed Heterogeneous Platforms
- Sustained Performance
- NUMA & Cache Aware Scheduling
- State-of-the-art Algorithms
- Capacity Level Scalability
- Performance Portability
- Implicit Communication
- Communication Overlapping



Graphic from icl.cs.utk.edu

Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J. "PaRSEC: Exploiting Heterogeneity to Enhance Scalability," IEEE Computing in Science and Engineering, Vol. 15, No. 6, 36-45, November, 2013.

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

Contents

2 pbdR

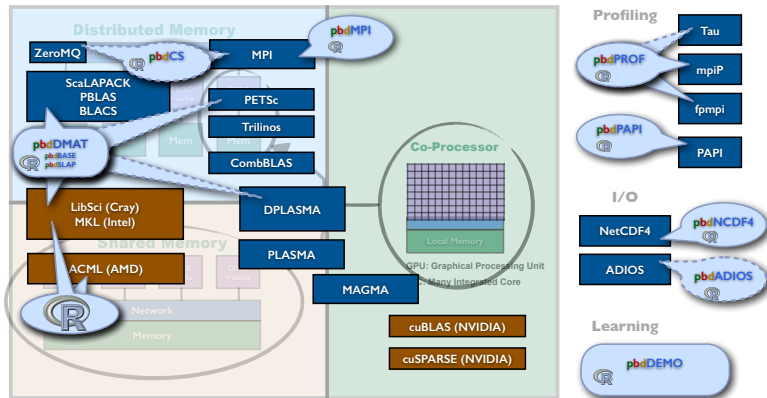
- The pbdR Project
- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis

2 pbdR

- The pbdR Project

- pbdMPI
- pbdDMAT
- RandSVD
- pbdMPI Example: Random Forest Prediction
- pbdMPI Example: Functional Data Analysis

pbdR Interfaces to Libraries: Sustainable Path



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**.

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

Integer? Not always obvious in R.

```
1 > is.integer(1)
2 [1] FALSE
3 > is.integer(2)
4 [1] FALSE
5 > is.integer(1:2)
6 [1] TRUE
```

pbdMPI lets R figure it out

Rmpi

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

pbdMPI

```
1 allreduce(x)
```

Single Program (SPMD): Runs Asynchronous Parallel

Rank Query Example

1_rank.r

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

Execute this batch script via:

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

Sample Output:

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

2 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

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

(a) 1×6

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

(b) 2×3

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

(c) 3×2

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

(d) 6×1

Table: Processor Grid Shapes with 6 Processors

2×3 block-cyclic grid on 6 processors: Global view “ddmatrix” class

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

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

2×3 block-cyclic grid on 6 processors: Local view “ddmatrix” class

$$\begin{array}{ccc}
 \left[\begin{array}{cc|cc} x_{11} & x_{12} & x_{17} & x_{18} \\ x_{21} & x_{22} & x_{27} & x_{28} \\ \hline x_{51} & x_{52} & x_{57} & x_{58} \\ x_{61} & x_{62} & x_{67} & x_{68} \\ \hline x_{91} & x_{92} & x_{97} & x_{98} \end{array} \right]_{5 \times 4} & \left[\begin{array}{cc|c} x_{13} & x_{14} & x_{19} \\ x_{23} & x_{24} & x_{29} \\ \hline x_{53} & x_{54} & x_{59} \\ x_{63} & x_{64} & x_{69} \\ \hline x_{93} & x_{94} & x_{99} \end{array} \right]_{5 \times 3} & \left[\begin{array}{cc} x_{15} & x_{16} \\ x_{25} & x_{26} \\ \hline x_{55} & x_{56} \\ x_{65} & x_{66} \\ \hline x_{95} & x_{96} \end{array} \right]_{5 \times 2} \\
 \left[\begin{array}{cc|cc} x_{31} & x_{32} & x_{37} & x_{38} \\ x_{41} & x_{42} & x_{47} & x_{48} \\ \hline x_{71} & x_{72} & x_{77} & x_{78} \\ x_{81} & x_{82} & x_{87} & x_{88} \end{array} \right]_{4 \times 4} & \left[\begin{array}{cc|c} x_{33} & x_{34} & x_{39} \\ x_{43} & x_{44} & x_{49} \\ \hline x_{73} & x_{74} & x_{79} \\ x_{83} & x_{84} & x_{89} \end{array} \right]_{4 \times 3} & \left[\begin{array}{cc} x_{35} & x_{36} \\ x_{45} & x_{46} \\ \hline x_{75} & x_{76} \\ x_{85} & x_{86} \end{array} \right]_{4 \times 2}
 \end{array}$$

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

pbdR Example Syntax

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

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

```
1 > x <- as.rowblock(x)
2 > x <- as.colblock(x)
3 > x <- redistribute(x, bldim=c(8, 8), ICTXT = 0)
```


pbdDMAT Scalability Benchmarks

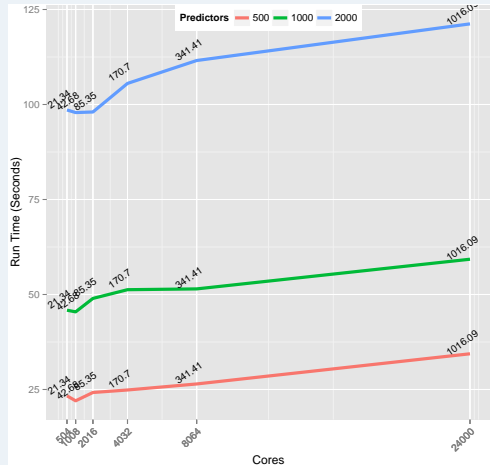
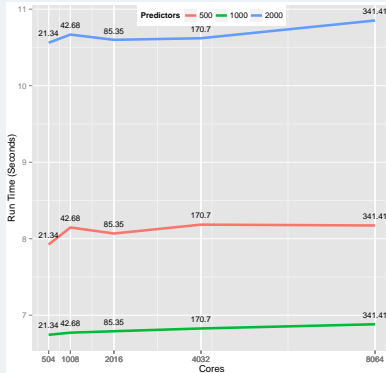
- Default choices throughout (no MKL, ACML, etc.)
- 1 core = 1 MPI process (Kraken: 6-core Opterons)
- Generate random matrix
 - Global Columns: 500, 1000, and 2000
 - Global Rows: fixed per core to make 43.4MiB
- Measure wall clock time
- “weak scaling” = global problem grows with core count



pbdDMAT Scalability Benchmarks

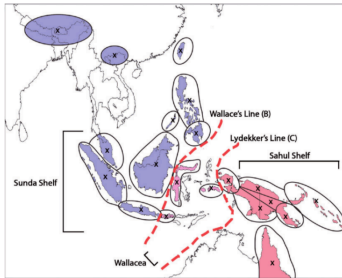
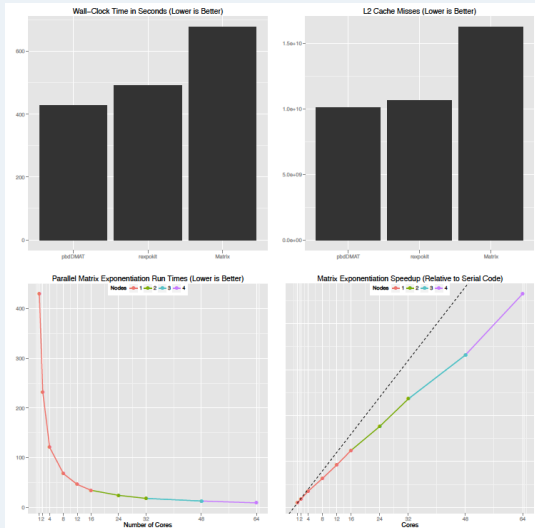
```
1 x <- ddmatrix("rnorm",
  nrow=n, ncol=p)
2 cov.x <- cov(x)
```

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



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



Schmidt and Matzke (2014) Distributed matrix exponentiation, The R User Conference (User! 2014), Los Angeles, CA, August 2014

2 pbdR

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

Randomized truncated SVD¹

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 Σ is nonnegative and diagonal.

Stage A:

- 1 Generate an $n \times 2k$ Gaussian test matrix Ω .
- 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: $B = \tilde{U}\Sigma V^*$.
- 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 ℓ 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 Ω .
- 2 Form $Y_0 = A\Omega$ and compute its QR factorization $Y_0 = Q_0 R_0$.
- 3 **for** $j = 1, 2, \dots, q$
- 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 $Y_j = A\tilde{Q}_j$ and compute its QR factorization $Y_j = Q_j R_j$.
- 6 **end**
- 7 $Q = Q_q$.

Serial R

```

1 randSVD <- 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 }
```

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

Randomized truncated SVD

Serial R

```

1 randSVD <- 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 }

```

Parallel pbdR

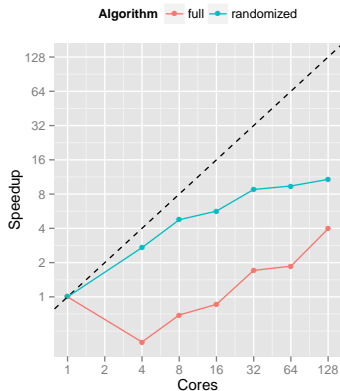
```

1 randSVD <- 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 }

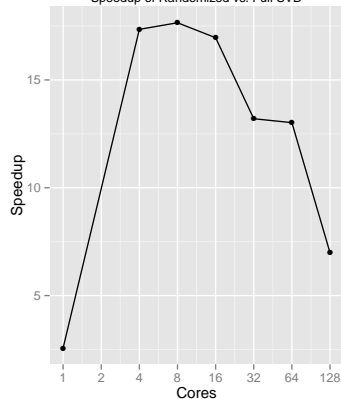
```

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

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



30 Singular Vectors from a 100,000 by 1,000 Matrix
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** ($20,000 \times 17$)



```

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 1: Random Forest Algorithm

- ➊ Build simple regression trees from random subsets of columns
- ➋ Use model averaging for prediction
- ➌ Package **randomForest** has a `combine()` function that enables the following parallel approach:
 - ➊ Everyone gets the same training data
 - ➋ Split regression tree building among processors (**randomForest**)
 - ➌ Use `allgather` to bring built predictors to all
 - ➍ Everyone combine predictors
 - ➎ Split prediction work by blocks of rows
 - ➏ Use `allreduce` to assess prediction
- ➍ 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

```
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(letter ~ ., train, ntree=500,
13                        norm.votes=FALSE)
14
15 ## predict test data
16 pred <- predict(rf.all, test)
17 correct <- sum(pred == test$letter)
18 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_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(),
13   norm.votes=FALSE)
14
15 rf.all <- do.call(combine, allgather(my.rf))
16
17 pred <- predict(rf.all, test)
18 correct <- allreduce(sum(pred == test$lettr))
19 comm.cat("Proportion Correct:", correct/(n_test), "\n")
```

Runs serial or on any number of cores

```
1 [beacon-login2 stats]$ time Rscript 4_rf_s.r
2 Proportion Correct: 0.96725
3 real 0m49.028s    user 0m48.626s    sys 0m0.335s
4 [beacon-login2 stats]$ time Rscript 4_rf_p.r
5 Proportion Correct: 0.96425
6 real 0m52.634s    user 0m51.914s    sys 0m0.598s
7 [beacon-login2 stats]$ time mpirun -np 2 Rscript 4_rf_p.r
8 Proportion Correct: 0.96425
9 real 0m28.349s    user 0m54.570s    sys 0m1.070s
10 [beacon-login2 stats]$ time mpirun -np 4 Rscript 4_rf_p.r
11 Proportion Correct: 0.963
12 real 0m16.380s    user 1m1.559s     sys 0m1.664s
13 [beacon-login2 stats]$ time mpirun -np 8 Rscript 4_rf_p.r
14 Proportion Correct: 0.963
15 real 0m11.010s    user 1m19.301s    sys 0m3.421s
16 [beacon-login2 stats]$ time mpirun -np 16 Rscript 4_rf_p.r
17 Proportion Correct: 0.9635
18 real 0m10.655s    user 2m32.508s    sys 0m6.624s
19 [beacon-login2 stats]$ time mpirun -np 32 Rscript 4_rf_p.r
20 Proportion Correct: 0.96325
21 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

fda.usc Package

Profiling min.basis()

```

1 > summaryRprof()
2 $by.total
3
4      total.time  total.pct  self.time  self.pct
5 "min.basis"    12.32    100.00      0.00     0.00
6 "type.CV"      6.54     53.08      0.02     0.16
7 "S.basis"      5.76     46.75      0.00     0.00
8 "drop"         4.20     34.09      0.00     0.00
9 "norm.fdata"   4.20     34.09      0.00     0.00
10 "metric"       4.18     33.93      1.04     8.44
11 "%*%"         3.98     32.31      3.98    32.31
12 "getbasispenalty" 2.72     22.08      0.02     0.16
13 "bsplinepen"   2.68     21.75      0.36     2.92
14 "int.simpson2" 2.54     20.62      1.96    15.91
15 "t"            2.10     17.05      0.10     0.81
16 "ppBspline"    1.60     12.99      0.82     6.66
17 . . .

```

Example: min.basis() 110 lines

SPMD: Add 5, change 3

```
1 min.basis <- function(fdataobj, type.CV = GCV.S, . . ., ...)
2 {
3   . . . 13 lines
4   library(pbdMPI)
5   init()
6   my.k <- get.jid(lenlambda)
7   my.gcv <- array(Inf, dim = c(lenbasis, length(my.k)))
8   . . . 36 lines
9   for (i in 1:lenbasis) {
10     . . . 3 lines
11     for (k in my.k) {
12       S2 <- S.basis(tt, base, lambda[k])
13       my.gcv[i, k - my.k[1] + 1] <-
14         type.CV(fdataobj, S = S2, W = W, trim =
15           par.CV$trim, draw = par.CV$draw, ...)
16     }
17   }
18   gcv <- do.call(cbind, allgather(my.gcv))
19   finalize()
20   . . . 48 lines
```


Contents

3 pbdCS

- Client-Server Demo

3 pbdCS

- Client-Server Demo

OMQ

Some explanation goes here The demo goes here

4 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/>
- **pbdDEMO** vignette
- [Googlegroup:RBigDataProgramming](#)