pbdMPI   GBD   Stats eg's   DMAT   pbdDMAT eg's   Wrapup
○○○○   ○○○○   ○○○○   ○○○○○○○○   ○○○
○○○○○○   ○○○   ○○○   ○○○○○○   ○○○○○
○○○○○   ○○○   ○○○○○   ○○○○○○○○

Programming with Big Data in R

Drew Schmidt[1] and George Ostrouchov[1,2]

[1]University of Tennessee, Knoxville, TN
[2]Oak Ridge National Laboratory, Oak Ridge, TN

OLCF Workshop
on Processing and Analysis of Very Large Data Sets
August 8, 2013

About This Presentation

Downloads

This presentation and supplemental materials are available at:

http://r-pbd.org/tutorial

Sample R scripts and pbs job scripts available on Chester:

/lustre/scratch/sw/r/3.0.1.new/chester/gnu4.7.3/
EXAMPLES/scripts.tar.gz

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
oooo	oooo	oooo	oooooooo	ooo	
oooooo	ooo	ooo	oooooo	ooooo	
ooooo	ooo	ooooo	oooooooo		

About This Presentation

Speaking Serial R with a Parallel Accent

The content of this presentation is based in part on the
pbdDEMO vignette *Speaking Serial R with a Parallel Accent*

http://goo.gl/HZkRt

It contains more examples, and sometimes added detail.

About This Presentation

Installation Instructions

Installation instructions for setting up a pbdR environment are available:

> http://r-pbd.org/install.html

This includes instructions for installing R, MPI, and pbdR.

Contents

Contents

Message Passing Interface (MPI)

- *MPI*: Standard for managing communications (data and instructions) between different nodes/computers.
- *Implementations*: OpenMPI, MPICH2, Cray MPT, ...
- Enables parallelism (via communication) on distributed machines.
- *Communicator*: manages communications between processors.

MPI Operations (1 of 2)

- **Managing a Communicator**: Create and destroy communicators.
  init() — initialize communicator
  finalize() — shut down communicator(s)

- **Rank query**: determine the processor's position in the communicator.
  comm.rank() — "who am I?"
  comm.size() — "how many of us are there?"

- **Printing**: Printing output from various ranks.
  comm.print(x)
  comm.cat(x)
  WARNING: only use these functions on *results*, never on yet-to-be-computed things.

Quick Example 1

Rank Query: 1_rank.r

```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 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
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○●	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

Managing a Communicator

Quick Example 2

Hello World: 2_hello.r

```
1  library(pbdMPI, quiet=TRUE)
2  init()
3
4  comm.print("Hello, world")
5
6  comm.print("Hello again", all.rank=TRUE, quiet=TRUE)
7
8  finalize()
```

Execute this script via:

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

Sample Output:

```
1  COMM.RANK = 0
2  [1] "Hello, world"
3  [1] "Hello again"
4  [1] "Hello again"
```

MPI Operations

1. Reduce
2. Gather
3. Broadcast
4. Barrier

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Reduce, Gather, Broadcast, and Barrier

Reductions — Combine results into single result

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
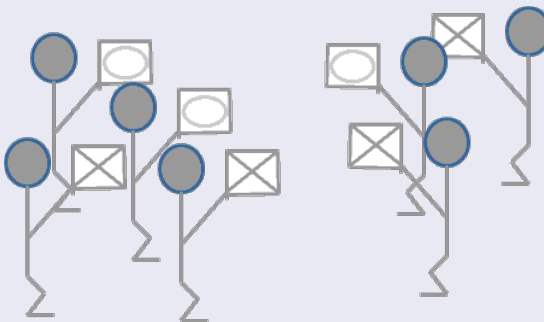
Reduce, Gather, Broadcast, and Barrier

Gather — Many-to-one

pbdMPI | GBD | Stats eg's | DMAT | pbdDMAT eg's | Wrapup
0000 | 0000 | 0000 | 00000000 | 000 |
0000●00 | 000 | 000 | 000000 | 00000 |
00000 | | 00000 | 00000000 | |

Reduce, Gather, Broadcast, and Barrier

Broadcast — One-to-many

pbdMPI   GBD   Stats eg's   DMAT   pbdDMAT eg's   Wrapup
○○○○   ○○○○   ○○○○   ○○○○○○○○○   ○○○
○○○○○●○   ○○○   ○○○   ○○○○○○○   ○○○○○
○○○○○   ○○○   ○○○○○   ○○○○○○○○○

Reduce, Gather, Broadcast, and Barrier
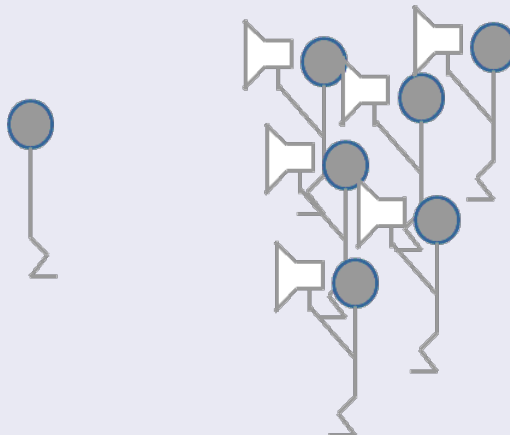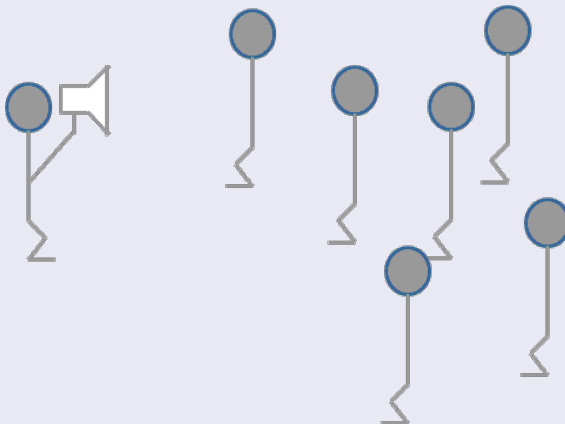
Barrier — Synchronization

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
0000	0000	0000	000000000	000	
000000●	000	000	000000	00000	
00000	000	00000	00000000		

Reduce, Gather, Broadcast, and Barrier
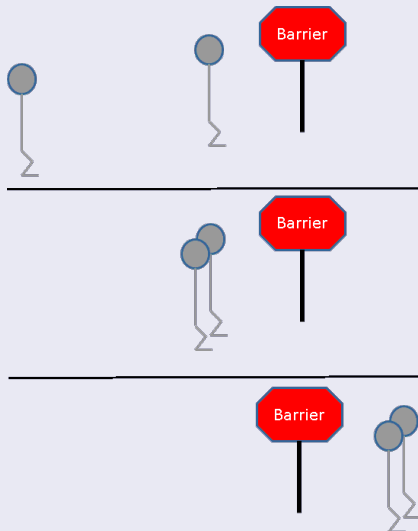
MPI Operations (2 of 2)

- **Reduction**: each processor has a number x; add all of them up, find the largest/smallest, ...
  reduce(x, op='sum') — reduce to one
  allreduce(x, op='sum') — reduce to all

- **Gather**: each processor has a number; create a new object on some processor containing all of those numbers.
  gather(x) — gather to one
  allgather(x) — gather to all

- **Broadcast**: one processor has a number x that every other processor should also have.
  bcast(x)

- **Barrier**: "computation wall"; no processor can proceed until *all* processors can proceed.
  barrier()

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
●○○○○	○○○	○○○○○	○○○○○○○○		

Other pbdMPI Tools

MPI Package Controls

The .SPMD.CT object allows for setting different package options with **pbdMPI**. See the entry *SPMD Control* of the **pbdMPI** manual for information about the .SPMD.CT object:

http://cran.r-project.org/web/packages/pbdMPI/pbdMPI.pdf

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○●○○○	○○○	○○○○○	○○○○○○○○		

Other pbdMPI Tools

Random Seeds

pbdMPI offers a simple interface for managing random seeds:

- `comm.set.seed(diff=TRUE)` — Independent streams via the **rlecuyer** package.
- `comm.set.seed(seed=1234, diff=FALSE)` — All processors use the same seed `seed=1234`
- `comm.set.seed(diff=FALSE)` — All processors use the same seed, determined by processor 0 (using the system clock and PID of processor 0).

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○●○○	○○○	○○○○○	○○○○○○○○		

Other pbdMPI Tools

Other Helper Tools

pbdMPI Also contains useful tools for Manager/Worker and task parallelism codes:

- **Task Subsetting**: Distributing a list of jobs/tasks
  `get.jid(n)`
- ***ply**: Functions in the *ply family.
  `pbdApply(X, MARGIN, FUN, ...)` — analogue of `apply()`
  `pbdLapply(X, FUN, ...)` — analogue of `lapply()`
  `pbdSapply(X, FUN, ...)` — analogue of `sapply()`

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○●○	○○○	○○○○○	○○○○○○○○		

Other pbdMPI Tools

Quick Comments for Using pbdMPI

1. Start by loading the package:

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

2. Always initialize before starting and finalize when finished:

```
1  init()
2
3  # ...
4
5  finalize()
```

Basic MPI Exercises

1. Experiment with Quick Examples 1 through 6, running them on 2, 4, and 8 processors.

Contents

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	●○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

The GBD Data Structure

Distributing Data

Problem: How to distribute the data

$$x = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \\ x_{4,1} & x_{4,2} & x_{4,3} \\ x_{5,1} & x_{5,2} & x_{5,3} \\ x_{6,1} & x_{6,2} & x_{6,3} \\ x_{7,1} & x_{7,2} & x_{7,3} \\ x_{8,1} & x_{8,2} & x_{8,3} \\ x_{9,1} & x_{9,2} & x_{9,3} \\ x_{10,1} & x_{10,2} & x_{10,3} \end{bmatrix}_{10\times3}$$

?

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○●○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

The GBD Data Structure

Distributing a Matrix Across 4 Processors: Block Distribution

Data

$$
x = \begin{bmatrix}
x_{1,1} & x_{1,2} & x_{1,3} \\
x_{2,1} & x_{2,2} & x_{2,3} \\
x_{3,1} & x_{3,2} & x_{3,3} \\
\hline
x_{4,1} & x_{4,2} & x_{4,3} \\
x_{5,1} & x_{5,2} & x_{5,3} \\
x_{6,1} & x_{6,2} & x_{6,3} \\
\hline
x_{7,1} & x_{7,2} & x_{7,3} \\
x_{8,1} & x_{8,2} & x_{8,3} \\
x_{9,1} & x_{9,2} & x_{9,3} \\
\hline
x_{10,1} & x_{10,2} & x_{10,3}
\end{bmatrix}_{10 \times 3}
$$

Processors

0
1
2
3

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○●	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

The GBD Data Structure

Distributing a Matrix Across 4 Processors: Local Load Balance

Data

Processors

$$x = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \\ x_{4,1} & x_{4,2} & x_{4,3} \\ x_{5,1} & x_{5,2} & x_{5,3} \\ x_{6,1} & x_{6,2} & x_{6,3} \\ x_{7,1} & x_{7,2} & x_{7,3} \\ x_{8,1} & x_{8,2} & x_{8,3} \\ x_{9,1} & x_{9,2} & x_{9,3} \\ x_{10,1} & x_{10,2} & x_{10,3} \end{bmatrix}_{10 \times 3}$$

0
1
2
3

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

The GBD Data Structure

The GBD Data Structure

Throughout the examples, we will make use of the Generalized Block Distribution, or GBD distributed matrix structure.

1. GBD is *distributed*. No processor owns all the data.

2. GBD is *non-overlapping*. Rows uniquely assigned to processors.

3. GBD is *row-contiguous*. If a processor owns one element of a row, it owns the entire row.

4. GBD is globally *row-major*, locally *column-major*.

5. GBD is often *locally balanced*, where each processor owns (almost) the same amount of data. But this is not required.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \\ x_{4,1} & x_{4,2} & x_{4,3} \\ x_{5,1} & x_{5,2} & x_{5,3} \\ x_{6,1} & x_{6,2} & x_{6,3} \\ x_{7,1} & x_{7,2} & x_{7,3} \\ x_{8,1} & x_{8,2} & x_{8,3} \\ x_{9,1} & x_{9,2} & x_{9,3} \\ x_{10,1} & x_{10,2} & x_{10,3} \end{bmatrix}$$

6. The last row of the local storage of a processor is adjacent (by global row) to the first row of the local storage of next processor (by communicator number) that owns data.

7. GBD is (relatively) easy to understand, but can lead to bottlenecks if you have many more columns than rows.

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	●○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

GBD: Example 1

Understanding GBD: Global Matrix

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

Processors = 0 1 2 3 4 5

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○●	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

GBD: Example 1

Understanding GBD: Load Balanced GBD

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

Processors =   0   1   2   3   4   5

Understanding GBD: Local View

$$
\left[
\begin{array}{ccccccccc}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29}
\end{array}
\right]_{2 \times 9}
$$

$$
\left[
\begin{array}{ccccccccc}
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49}
\end{array}
\right]_{2 \times 9}
$$

$$
\left[
\begin{array}{ccccccccc}
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69}
\end{array}
\right]_{2 \times 9}
$$

$$
\left[
\begin{array}{ccccccccc}
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79}
\end{array}
\right]_{1 \times 9}
$$

$$
\left[
\begin{array}{ccccccccc}
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89}
\end{array}
\right]_{1 \times 9}
$$

$$
\left[
\begin{array}{ccccccccc}
x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99}
\end{array}
\right]_{1 \times 9}
$$

Processors = 0 1 2 3 4 5

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	●○○	○○○○○	○○○○○○○○		

GBD: Example 2

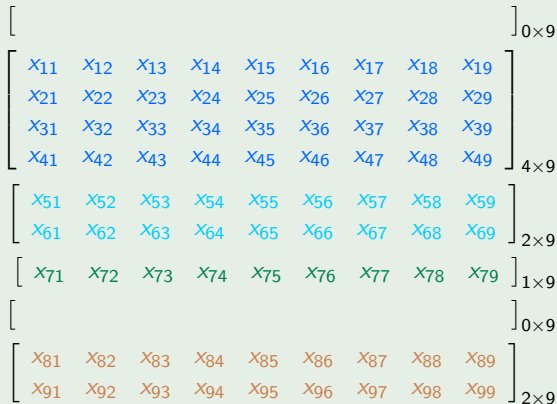## Understanding GBD: Non-Balanced GBD

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

Processors = 0 1 2 3 4 5

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○●○	○○○○○	○○○○○○○○○		

GBD: Example 2

Understanding GBD: Local View

$$
\begin{bmatrix} & & & & & & & & \end{bmatrix}_{0\times 9}
$$

$$
\begin{bmatrix}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49}
\end{bmatrix}_{4\times 9}
$$

$$
\begin{bmatrix}
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69}
\end{bmatrix}_{2\times 9}
$$

$$
\begin{bmatrix}
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79}
\end{bmatrix}_{1\times 9}
$$

$$
\begin{bmatrix} & & & & & & & & \end{bmatrix}_{0\times 9}
$$

$$
\begin{bmatrix}
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\
x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99}
\end{bmatrix}_{2\times 9}
$$

Processors =   0   1   2   3   4   5

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○●	○○○○○	○○○○○○○○		

GBD: Example 2

Quick Comment for GBD

Local pieces of GBD distributed objects will be given the suffix `.gbd` to visually help distinguish them from global objects. This suffix carries no semantic meaning.
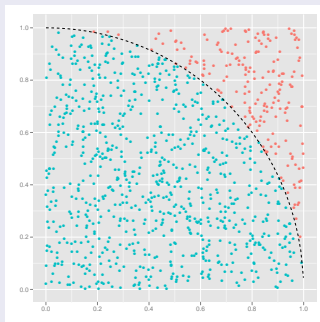
Contents

3. Basic Statistics Examples
   - pbdMPI Example: Monte Carlo Simulation
   - pbdMPI Example: Sample Covariance
   - pbdMPI Example: Linear Regression

Example 1: Monte Carlo Simulation

Sample N uniform observations (x_i, y_i) in the unit square $[0,1] \times [0,1]$. Then

$$\pi \approx 4 \left(\frac{\# \ Inside \ Circle}{\# \ Total} \right) = 4 \left(\frac{\# \ \text{Blue}}{\# \ \text{Blue} + \# \ \text{Red}} \right)$$

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○●○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

pbdMPI Example: Monte Carlo Simulation

Example 1: Monte Carlo Simulation GBD Algorithm

1. Let n be big-ish; we'll take $n = 50,000$.

2. Generate an $n \times 2$ matrix x of standard uniform observations.

3. Count the number of rows satisfying $x^2 + y^2 \leq 1$

4. Ask everyone else what their answer is; sum it all up.

5. Take this new answer, multiply by 4 and divide by n

6. If my rank is 0, print the result.

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○●○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

pbdMPI Example: Monte Carlo Simulation

Example 1: Monte Carlo Simulation Code

Serial Code

```
1 N <- 50000
2 X <- matrix(runif(N * 2), ncol=2)
3 r <- sum(rowSums(X^2) <= 1)
4 PI <- 4*r/N
5 print(PI)
```

Parallel Code

```
1  library(pbdMPI, quiet = TRUE)
2  init()
3  comm.set.seed(diff=TRUE)
4
5  N.gbd <- 50000 / comm.size()
6  X.gbd <- matrix(runif(N.gbd * 2), ncol = 2)
7  r.gbd <- sum(rowSums(X.gbd^2) <= 1)
8  r <- allreduce(r.gbd)
9  PI <- 4*r/(N.gbd * comm.size())
10 comm.print(PI)
11
12 finalize()
```

pbdMPI	GBD	**Stats eg's**	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○●	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

pbdMPI Example: Monte Carlo Simulation

Note

For the remainder, we will exclude loading, init, and finalize calls.

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	●○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

pbdMPI Example: Sample Covariance

Example 2: Sample Covariance

$$cov(x_{n \times p}) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_x)(x_i - \mu_x)^T$$

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○●○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

pbdMPI Example: Sample Covariance

Example 2: Sample Covariance GBD Algorithm

1. Determine the total number of rows N.

2. Compute the vector of column means of the full matrix.

3. Subtract each column's mean from that column's entries in each local matrix.

4. Compute the crossproduct locally and reduce.

5. Divide by $N - 1$.

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○●	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

pbdMPI Example: Sample Covariance

Example 2: Sample Covariance Code

Serial Code

```
1  N <- nrow(X)
2  mu <- colSums(X) / N
3
4  X <- sweep(X, STATS=mu, MARGIN=2)
5  Cov.X <- crossprod(X) / (N-1)
6
7  print(Cov.X)
```

Parallel Code

```
1  N <- allreduce(nrow(X.gbd), op="sum")
2  mu <- allreduce(colSums(X.gbd) / N, op="sum")
3
4  X.gbd <- sweep(X.gbd, STATS=mu, MARGIN=2)
5  Cov.X <- allreduce(crossprod(X.gbd), op="sum") / (N-1)
6
7  comm.print(Cov.X)
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	●○○○○	○○○○○○○○		

pbdMPI Example: Linear Regression

Example 3: Linear Regression

Find β such that

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

When $\mathbf{X}$ is full rank,

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
OOOO	OOOO	OOOO	OOOOOOOOO	OOO	
OOOOOO	OOO	OOO	OOOOOO	OOOOO	
OOOOO	OOO	O●OOO	OOOOOOOO		

pbdMPI Example: Linear Regression

Example 3: Linear Regression GBD Algorithm

1. Locally, compute $tx = x^T$

2. Locally, compute $A = tx * x$. Query every other processor for this result and sum up all the results.

3. Locally, compute $B = tx * y$. Query every other processor for this result and sum up all the results.

4. Locally, compute $A^{-1} * B$

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○●○○	○○○○○○○○		

pbdMPI Example: Linear Regression

Example 3: Linear Regression Code

Serial Code

```r
tX <- t(X)
A <- tX %*% X
B <- tX %*% y

ols <- solve(A) %*% B
```

Parallel Code

```r
tX.gbd <- t(X.gbd)
A <- allreduce(tX.gbd %*% X.gbd, op = "sum")
B <- allreduce(tX.gbd %*% y.gbd, op = "sum")

ols <- solve(A) %*% B
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

pbdMPI Example: Linear Regression

MPI Exercises

1. Experiment with Statistics Examples 1 through 3, running them on 2, 4, and 8 processors.

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

pbdMPI Example: Linear Regression

Advanced MPI Exercises I

1. Write a script that will have each processor randomly take a sample of size 1 of TRUE and FALSE. Have each processor print its result.

2. Modify the script in Exercise 1 above to determine if any processors sampled TRUE. Do the same to determine if all processors sampled TRUE. In each case, print the result. Compare to the functions comm.all() and comm.any().

3. Generate 50,000,000 (total) random normal values in parallel on 2, 4, and 8 processors. Time each run.

Advanced MPI Exercises II

④ Distribute the matrix `x <- matrix(1:24, nrow=12)` in GBD format across 4 processors and call it `x.spmd`.

    ① Add `x.spmd` to itself.

    ② Compute the mean of `x.spmd`.

    ③ Compute the column means of `x.spmd`.

Contents

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	●○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

Introduction to Distributed Matrices

Distributed Matrices

Most problems in data science are matrix algebra problems, so:

$$\text{Distributed matrices} \implies \text{Handle Bigger data}$$

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○●○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

Introduction to Distributed Matrices

Distributed Matrices

High level OOP allows *native* serial R syntax:

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

However. . .

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
oooo	oooo	oooo	oo●ooooo	ooo	
oooooo	ooo	ooo	oooooo	ooooo	
ooooo	ooo	ooooo	oooooooo		

Introduction to Distributed Matrices

Distributed Matrices

DMAT:

- **D**istributed **MAT**rix data structure.

- No single processor should hold all of the data.

- Block-cyclic matrix distributed across a 2-dimensional grid of processors.

- Very robust, but confusing data structure.

Distributed Matrices

(a) Block   (b) Cyclic   (c) Block-Cyclic

Figure: Matrix Distribution Schemes

Distributed Matrices
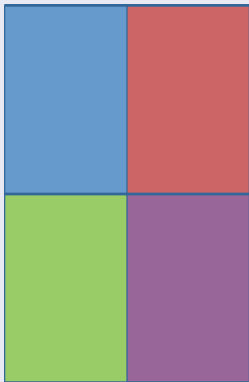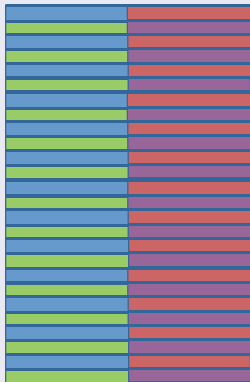
(a) 2d Block   (b) 2d Cyclic   (c) 2d Block-Cyclic
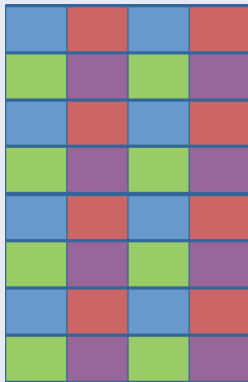
Figure: Matrix Distribution Schemes Onto a 2-Dimensional Grid

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
oooo	oooo	oooo	ooooo●oo	ooo	
oooooo	ooo	ooo	oooooooo	ooooo	
ooooo	ooo	ooooo	oooooooo		

Introduction to Distributed Matrices

Processor Grid Shapes

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

(a) 1×6 (b) 2×3 (c) 3×2 (d) 6×1

Table: Processor Grid Shapes with 6 Processors

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○●○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

Introduction to Distributed Matrices

Distributed Matrices

The data structure is a special R class (in the OOP sense) called
ddmatrix. It is the "under the rug" storage for a block-cyclic
matrix distributed onto a 2-dimensional processor grid.

$$
\text{ddmatrix} = \begin{cases}
\textbf{Data} & \text{S4 local submatrix, an R matrix} \\
\textbf{dim} & \text{S4 dimension of the global matrix, a numeric pair} \\
\textbf{ldim} & \text{S4 dimension of the local submatrix, a numeric pair} \\
\textbf{bldim} & \text{S4 ScaLAPACK blocking factor, a numeric pair} \\
\textbf{CTXT} & \text{S4 BLACS context, an numeric singleton}
\end{cases}
$$

with prototype

$$
\text{new("ddmatrix")} = \begin{cases}
\textbf{Data} & = \texttt{matrix(0.0)} \\
\textbf{dim} & = \texttt{c(1,1)} \\
\textbf{ldim} & = \texttt{c(1,1)} \\
\textbf{bldim} & = \texttt{c(1,1)} \\
\textbf{CTXT} & = \texttt{0.0}
\end{cases}
$$

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○●	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

Introduction to Distributed Matrices

Distributed Matrices: The Data Structure

Example: an 9×9 matrix is distributed with a "block-cycling" factor of 2×2 on a 2×2 processor grid:
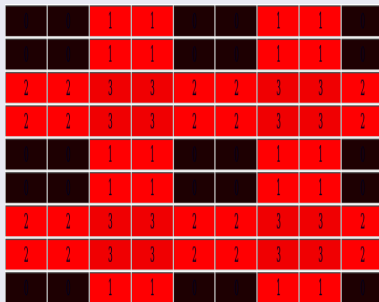
$$= \begin{cases} \textbf{Data} & = \texttt{matrix}(\dots) \\ \textbf{dim} & = \texttt{c(9, 9)} \\ \textbf{ldim} & = \texttt{c(}\dots\texttt{)} \\ \textbf{bldim} & = \texttt{c(2, 2)} \\ \textbf{CTXT} & = 0 \end{cases}$$

See http://acts.nersc.gov/scalapack/hands-on/datadist.html

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	●○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

DMAT Distributions

Understanding Dmat: Global Matrix

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

DMAT: 1-dimensional Row Block

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

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

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup

DMAT Distributions

DMAT: 2-dimensional Row Block

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

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

DMAT: 1-dimensional Row Cyclic

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

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

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○●○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○○		

DMAT Distributions

DMAT: 2-dimensional Row Cyclic

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

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

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup

DMAT Distributions

DMAT: 2-dimensional Block-Cyclic

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\ x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\ x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\ x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{bmatrix}_{9\times9}$$

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

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	●○○○○○○○		

pbdDMAT

The `DMAT` Data Structure

The more complicated the processor grid, the more complicated the distribution.

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
oooo	oooo	oooo	oooooooooo	ooo	
oooooo	ooo	ooo	oooooooo	ooooo	
ooooo	ooo	ooooo	oo●ooooooo		

pbdDMAT

DMAT: 2-dimensional Block-Cyclic with 6 Processors

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

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

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○●○○○○○○		

pbdDMAT

Understanding DMAT: Local View

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

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

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

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○●○○○○○		

pbdDMAT

The `DMAT` Data Structure

1 `DMAT` is *distributed*. No one processor owns all of the matrix.

2 `DMAT` is *non-overlapping*. Any piece owned by one processor is owned by no other processors.

3 `DMAT` can be row-contiguous or not, depending on the processor grid and blocking factor used.

4 `DMAT` is locally column-major and globally, it depends. . .

$$
\begin{bmatrix}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} \\
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} \\
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} \\
x_{91} & x_{92} & x_{93} & x_{94} & x_{95}
\end{bmatrix}
$$

6 `GBD` is a generalization of the one-dimensional block `DMAT` distribution. Otherwise there is no relation.

7 `DMAT` is confusing, but very robust.

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
oooo	oooo	oooo	ooooooooo	ooo	
oooooo	ooo	ooo	ooooooo	ooooo	
ooooo	ooo	ooooo	ooooo●ooo		

pbdDMAT

Pros and Cons of This Data Structure

Pros
- Fast for distributed matrix computations

Cons
- Literally everything else

This is why we hide most of the distributed details.

The details are there if you want them (you don't want them).

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
0000	0000	0000	00000000	000	
000000	000	000	000000	00000	
00000	000	00000	00000●00		

pbdMAT

Distributed Matrix Methods

pbdDMAT has over 100 methods with *identical* syntax to R:

- `` `[` ``, rbind(), cbind(), ...
- lm.fit(), prcomp(), cov(), ...
- `` `%*%` ``, solve(), svd(), norm(), ...
- median(), mean(), rowSums(), ...

Serial Code

```
1  cov(x)
```

Parallel Code

```
1  cov(x)
```

Comparing pbdMPI and pbdDMAT

pbdMPI:

- MPI + sugar.
- GBD not the only structure **pbdMPI** can handle (just a useful convention).

pbdDMAT:

- More of a software package.
- DMAT structure *must* be used for **pbdDMAT**.
- If the data is not 2d block-cyclic compatible, DMAT will *definitely* give the wrong answer.

pbdMPI	GBD	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○●		

pbdDMAT

Quick Comments for Using pbdDMAT

1 Start by loading the package:

```
1  library(pbdDMAT, quiet = TRUE)
```

2 Always initialize before starting and finalize when finished:

```
1  init.grid()
2
3  # ...
4
5  finalize()
```

3 Distributed DMAT objects will be given the suffix .dmat to visually help distinguish them from global objects. This suffix carries no semantic meaning.

Contents

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
oooo	oooo	oooo	oooooooo	●oo	
oooooo	ooo	ooo	oooooo	ooooo	
ooooo	ooo	ooooo	oooooooo		

Working with Distributed Matrices

Creating DMAT Objects

```
1  a.dmat <- as.ddmatrix(a)
2  b.dmat <- ddmatrix(0, nrow=100, ncol=100)
3  c.dmat <- diag(1, nrow=100, ncol=100, type='ddmatrix')
4  d.dmat <- ddmatrix("rnorm", nrow=100, ncol=100, mean=10,
        sd=100)
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
००००	००००	००००	०००००००००	●●●	
००००००	०००	०००	००००००००	०००००	
०००००	०००	०००००	०००००००००		

Working with Distributed Matrices

Extraction

```
1  x.dmat <- ddmatrix(1:100, nrow=10)
2  y.dmat <- x.dmat[1:8, 10:1]
3
4  y <- as.matrix(y.dmat)
5  comm.print(y)
```

```
1   COMM.RANK = 0
2         [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
3   [1,]    91   81   71   61   51   41   31   21   11     1
4   [2,]    92   82   72   62   52   42   32   22   12     2
5   [3,]    93   83   73   63   53   43   33   23   13     3
6   [4,]    94   84   74   64   54   44   34   24   14     4
7   [5,]    95   85   75   65   55   45   35   25   15     5
8   [6,]    96   86   76   66   56   46   36   26   16     6
9   [7,]    97   87   77   67   57   47   37   27   17     7
10  [8,]    98   88   78   68   58   48   38   28   18     8
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○○	○○●	
○○○○○○	○○○	○○○	○○○○○○	○○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○		

Working with Distributed Matrices

Other Operations

```
1  x.dmat <- ddmatrix(1:100, nrow=10)
2  y.dmat <- x.dmat + 1:10
3  z.dmat <- scale(y.dmat, center=TRUE, scale=FALSE)
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
0000	0000	0000	00000000	000	
000000	000	000	000000	●0000	
00000	000	00000	00000000		

Statistics Examples with pbdDMAT

Sample Covariance

Serial Code

```
1  Cov.X <- cov(X)
2  print(Cov.X)
```

Parallel Code

```
1  Cov.X <- cov(X)
2  print(Cov.X)
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○	○○○○	○○○○	○○○○○○○○	○○○	
○○○○○○	○○○	○○○	○○○○○○○	○●○○○○	
○○○○○	○○○	○○○○○	○○○○○○○○○		

Statistics Examples with pbdDMAT

Linear Regression

Serial Code

```
1  tX <- t(X)
2  A <- tX %*% X
3  B <- tX %*% y
4
5  ols <- solve(A) %*% B
6
7  # or
8  ols <- lm.fit(X, y)
```

Parallel Code

```
1  tX <- t(X)
2  A <- tX %*% X
3  B <- tX %*% y
4
5  ols <- solve(A) %*% B
6
7  # or
8  ols <- lm.fit(X, y)
```

pbdMPI	GBD	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
0000	0000	0000	00000000	000	
000000	000	000	000000	00●00	
00000	000	00000	00000000		

Statistics Examples with pbdDMAT

Distributed Matrices

pbdDEMO contains many other examples of reading and managing GBD and DMAT data

DMAT Exercises

1. Experiment with DMAT Examples 1 through 5, running them on 2 and 4 processors.

Advanced DMAT Exercises I

1. Subsetting, selection, and filtering are basic matrix operations featured in R. The following may look silly, but it is useful for data processing. Let `x.dmat <- ddmatrix(1:30, 10, 3)`. Do the following:

   - ```
     y.dmat <- x.dmat[c(1, 5, 4, 3), ]
     y.dmat <- x.dmat[c(10:3, 5, 5), ]
     y.dmat <- x.dmat[1:5, 3:1]
     ```

   - ```
     y.dmat <- x.dmat[x.dmat[, 2] > 13, ]
     y.dmat <- x.dmat[x.dmat[, 2] > x.dmat[, 3], ]
     y.dmat <- x.dmat[, x.dmat[2,] > x.dmat[3, ]]
     y.dmat <- x.dmat[c(1, 3, 5), x.dmat[, 2] >
     x.dmat[, 3]]
     ```

Advanced DMAT Exercises II

2. The method `crossprod()` is an optimized form of the crossproduct computation `t(x.dmat) %*% x.dmat`. For this exercise, let `x.dmat <- ddmatrix(1:30, nrow=10, ncol=3)`.

    1. Verify that these computations really do produce the same results.
    2. Time each operation. Which is faster?

3. The `prcomp()` method returns rotations for all components. Computationally verify by example that these rotations are orthogonal, i.e., that their crossproduct is the identity matrix.

Contents

pbdMPI
oooo
oooooo
ooooo

GBD
oooo
ooo
ooo

Stats eg's
oooo
ooo
ooooo

DMAT
oooooooo
oooooo
oooooooo

pbdDMAT eg's
ooo
ooooo

Wrapup

Where to Learn More

- Our website `http://r-pbd.org/`
- The **pbdDEMO** package
  `http://cran.r-project.org/web/packages/pbdDEMO/`
- The **pbdDEMO** Vignette: `http://goo.gl/HZkRt`
- Our Google Group: `http://group.r-pbd.org`

Thanks for coming!

Questions? Comments?