

# Programming with Big Data in R

Drew Schmidt and George Ostrouchov

July 8, 2013



## Affiliations and Support

The pbdR Core Team

<http://r-pbd.org>

Wei-Chen Chen<sup>1</sup>, George Ostrouchov<sup>1,2</sup>, Pragneshkumar Patel<sup>2</sup>, Drew Schmidt<sup>1</sup>

Ostrouchov, Patel, and Schmidt were supported in part by the project “NICS Remote Data Analysis and Visualization Center” funded by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center.

Chen and Ostrouchov were supported in part by the project “Visual Data Exploration and Analysis of Ultra-large Climate Data” funded by U.S. DOE Office of Science under Contract No. DE-AC05-00OR22725.

<sup>1</sup>Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN

<sup>2</sup>Remote Data Analysis and Visualization Center, University of Tennessee, Knoxville, TN

# About This Presentation

## Downloads

This presentation and supplemental materials are available at:

<http://r-pbd.org/user2013>

## 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*

<https://github.com/wrathematics/pbdDEMO/blob/master/inst/doc/pbdDEMO-guide.pdf?raw=true>

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.

# About This Presentation

## Conventions

We use:

- “.” as a decimal mark
- “,” as order of magnitude separator

Example	Yes	No
One million	1,000,000	1.000.000
One half	0.5	0,5
One thousand and one half	1,000.5	1.000,5

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo oooooooooooo oooooooooooo ooooo	oooo oooo oooo	oooo oooooooooo oooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

# Contents

- 1 Introduction
- 2 pbdR
- 3 Introduction to pbdMPI
- 4 The Generalized Block Distribution
- 5 Brief Intermission
- 6 Basic Statistics Examples
- 7 Introduction to pbdDMAT
- 8 Examples Using pbdDMAT
- 9 Wrapup

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
ooooooooooooo ooooooooooooo ooooo	oooo oooo	oooo ooooooooo ooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

# Contents

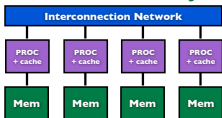
- 1 Introduction
  - Quick Overview of Parallel Hardware
  - A Concise Introduction to Parallelism
  - R and Parallelism



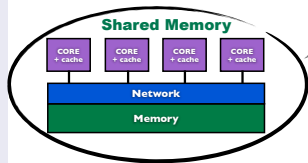


## Your Laptop or Desktop

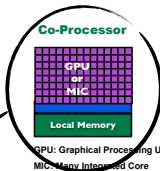
### Distributed Memory



### Shared Memory



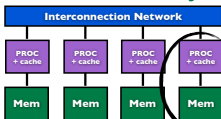
### Co-Processor



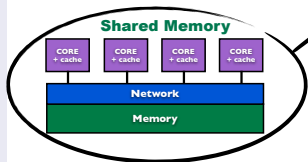
GPU: Graphical Processing Unit  
MIC: Many Integrated Core

## A Server or Cluster

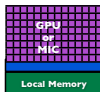
### Distributed Memory



### Shared Memory

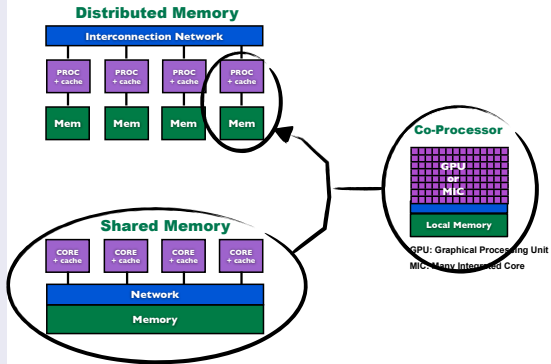


### Co-Processor

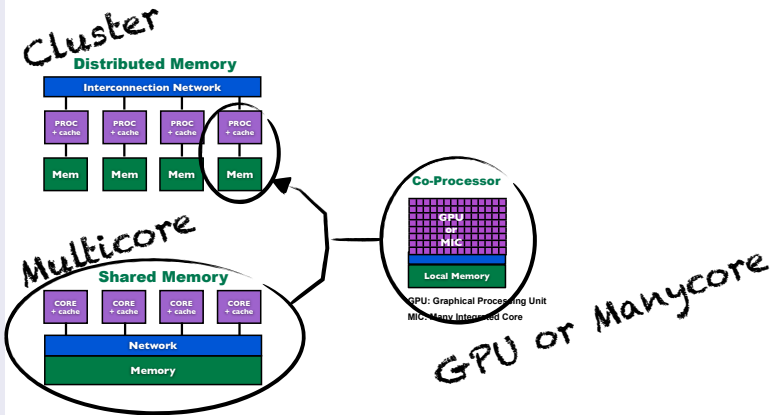


GPU: Graphical Processing Unit  
MIC: Many Integrated Core

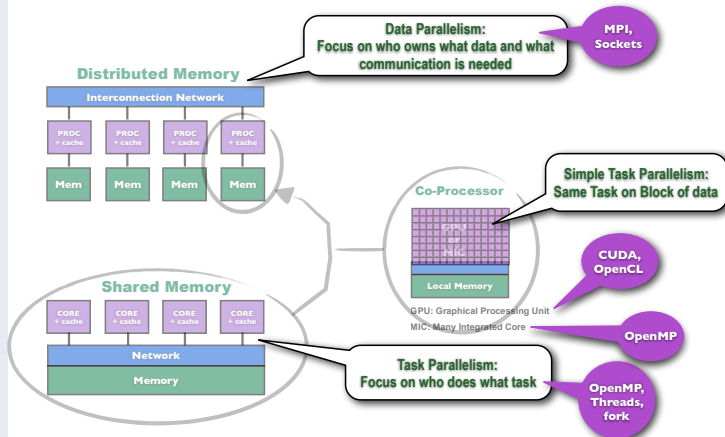
## Server to Supercomputer



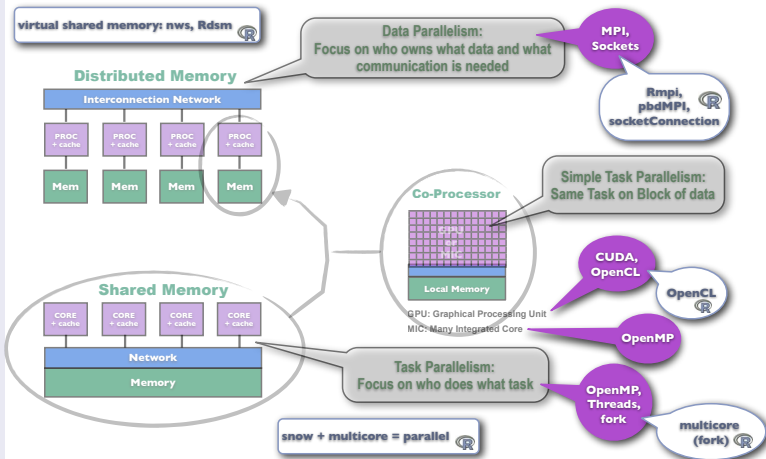
## Knowing the Right Words



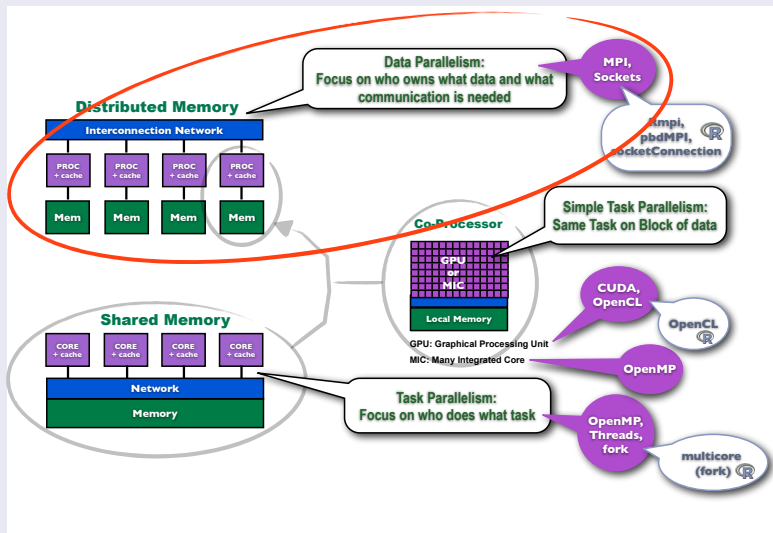
## "Native" Programming Models and Tools



## R Interfaces to Native Tools

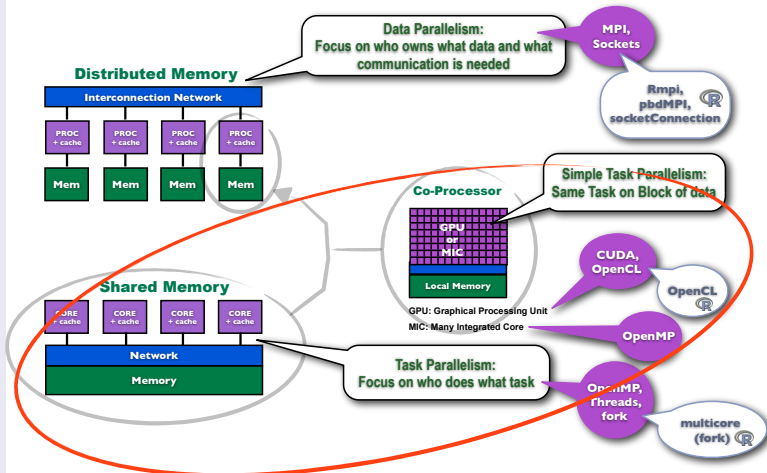


## 30+ Years of Parallel Computing Research

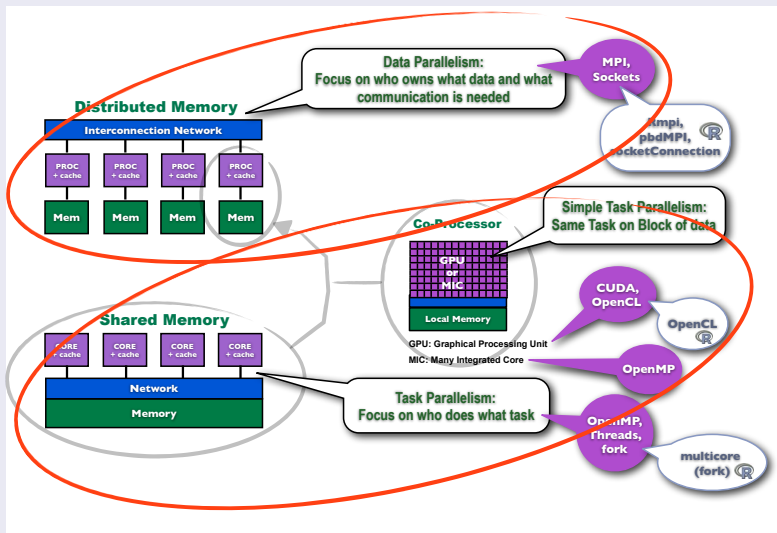




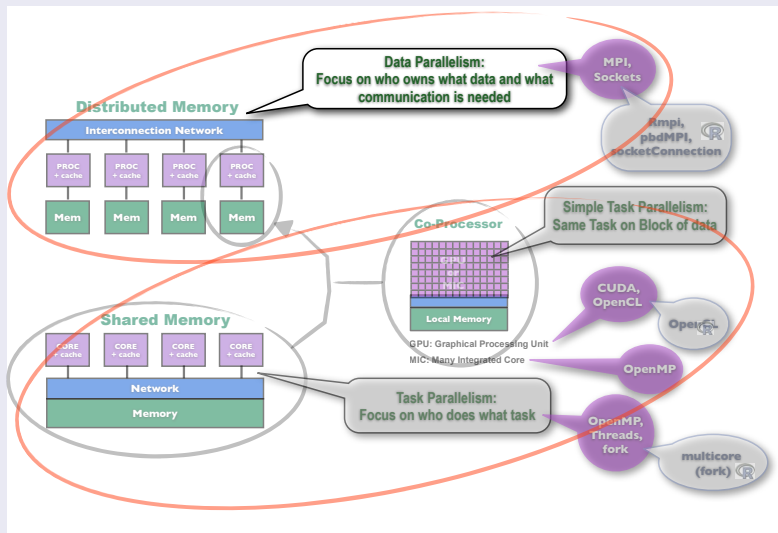
## Last 10 years of Advances



## Putting It All Together Challenge



## pbdR Focus on Data Parallelism



Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo	oooo	oooo	ooo		oooo	ooooo	ooo	
●oooooooooooo	oooo	oooooooooooo	ooo		ooo	oooooooo	ooooo	
ooooo		oooooo	ooo		ooo	oooooooooo	oo	

## A Concise Introduction to Parallelism

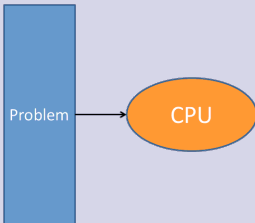
### What is Parallelism?

Broadly, *doing more than one thing at a time.*

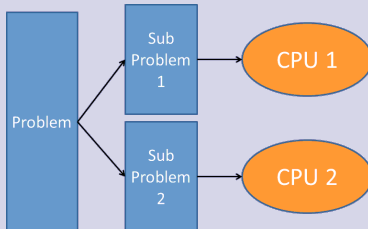
The simultaneous use of multiple compute resources to solve a computational problem:

## Parallelism

### Serial Programming

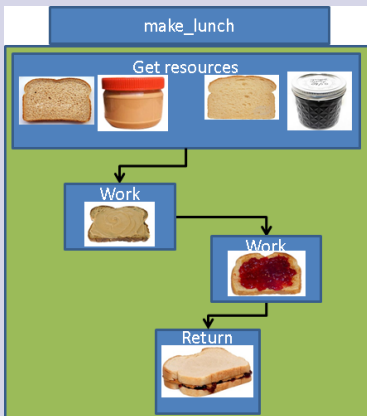


### Parallel Programming

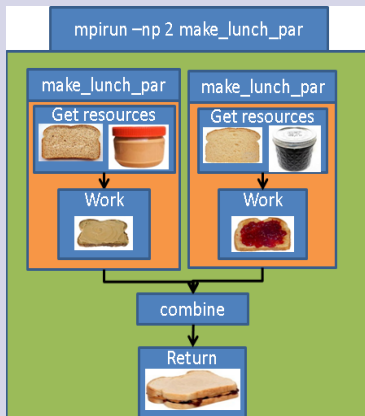


## Parallelism

### Serial Programming



### Parallel Programming



### Kinds of Parallelism

- *Data Parallelism*: Data is distributed
- *Task Parallelism*: Tasks are distributed

## pbdR Paradigms: Data Parallelism

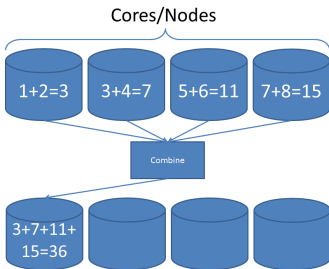
With data parallelism:

- No one processor/node owns all the data.
- Processors own local pieces of a (conceptually) global object

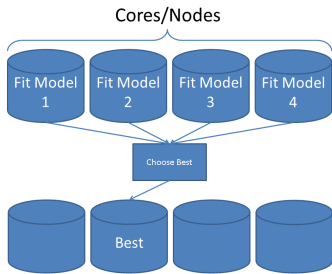


## Data vs Task Parallelism

### Data Parallelism



### Task Parallelism



Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo oooooooo●oooo ooooo	oooo oooo	oooo ooooooooo oooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

## A Concise Introduction to Parallelism

### Difficulty

- ① *Implicit parallelism*: Parallel details hidden from user
- ② *Explicit parallelism*: Some assembly required. . .
- ③ *Embarrassingly Parallel*: Also called *loosely coupled*. Obvious how to make parallel; lots of independence in computations.
- ④ *Tightly Coupled*: Opposite of embarrassingly parallel; lots of dependence in computations.

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo oooooooo●oooo ooooo	oooo oooo	oooo ooooooooo ooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

## A Concise Introduction to Parallelism

### Scalability

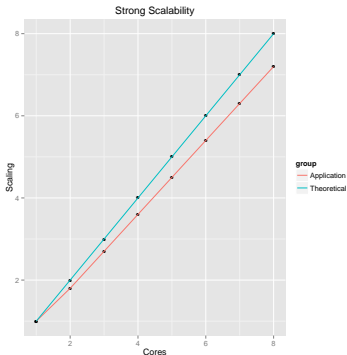
*Scalability*: unitless measure of performance;

$$\frac{\tau_i}{\tau_0}$$

## Types of Scalability: Strong and Weak

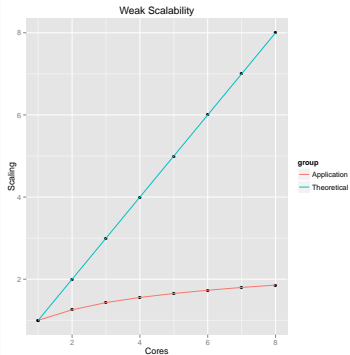
### Strong

Fix *total* data size



### Weak

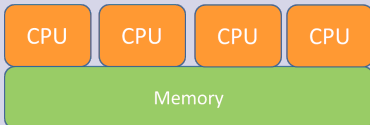
Fix *local* data size



## Shared and Distributed Memory Machines

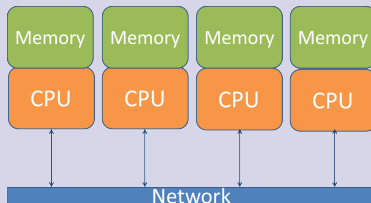
### Shared Memory

Different processors can directly access and modify each others' memory. There is only one node.



### Distributed

Different processors/nodes can not directly access/modify different processors'/nodes' memory.



## Shared and Distributed Memory Machines

### Shared Memory Machines

Thousands of cores



*Nautilus*, University of Tennessee

1024 cores

### Distributed Memory Machines

Hundreds of thousands of cores



*Kraken*, University of Tennessee

112,896 cores

## R and Parallelism

### What about R?

## Problems with Serial R

- 1 Slow.
- 2 If you don't know what you're doing, it's *really* slow.
- 3 Performance improvements usually for small machines.
- 4 Very ram intensive.
- 5 Chokes on big data.



## Parallel R Packages

### Shared Memory

- 1 **foreach**
- 2 **parallel**
- 3 **snow**
- 4 **multicore**

### Distributed

- 1 **Rmpi**
- 2 **R+Hadoop**
- 3 **pbdR**

## R and Parallelism

The solution to many of R's problems is parallelism. However ...

### What we have

- ① Mostly serial.
- ② Mostly not distributed
- ③ Data parallelism mostly explicit

### What we want

- ① Mostly parallel.
- ② Mostly distributed.
- ③ Mostly implicit.

## Why We Need Parallelism

- 1 Saves time (long term).
- 2 Data size is skyrocketing.
- 3 Necessary for many problems.
- 4 Like it or not, it's coming.
- 5 *It's really cool.*

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo	oooo	oooo	ooo		oooo	ooooo	ooo	
oooooooooooo	oooo	oooooooooo	ooo		ooo	oooooo	oooo	
ooooo		oooooo	ooo		ooo	oooooooo	oo	

# Contents

- 2 pbdR
  - The pbdR Project
  - pbdR Paradigms

## Programming with Big Data in R (pbdR)

Striving for *Productivity, Portability, Performance*

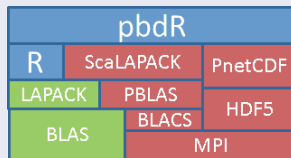
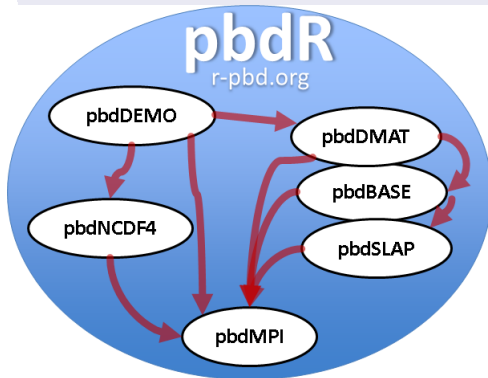


- *Free<sup>a</sup>* R packages.
- Bridging high-performance C with high-productivity of R
- Scalable, big data analytics.
- Distributed data details implicitly managed.
- Methods have syntax *identical* to R.
- Powered by state of the art numerical libraries (MPI, ScaLAPACK, ...)

---

<sup>a</sup>MPL, BSD, and GPL licensed

## pbdB Packages



## pbdR Packages — <http://code.r-pbd.org>

Released to CRAN:

- **pbdMPI**: MPI bindings (explicit, low-level)
- **pbdSLAP**: Foreign library (just install it, nothing to use)
- **pbdBASE**: Compiled code (used by DMAT, also for devs)
- **pbdDMAT**: Distributed matrices (mostly implicit, high-level)
- **pbdNCDF4**: Parallel NetCDF4 reader
- **pbdDEMO**: Package demonstrations, examples, vignette written in textbook style

Future Development:

- Updates and expansions
- Profiling Tools for Parallel Computing with R
- ...

## Example Syntax

```

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

```

Look familiar?

*The above runs on 1 core with R or 10,000 cores with pbdR*



## pbdR Paradigms

Programs that use pbdR are utilize:

- Batch execution
- Single Program/Multiple Data (SPMD) style

And generally utilize:

- Data Parallelism

## Batch Execution

- Non-interactive
- Use

```
1 Rscript my_script.r
```

or

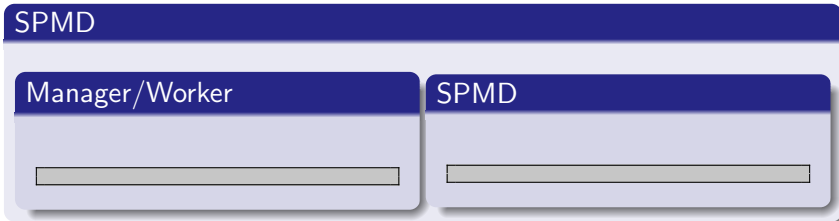
```
1 R CMD BATCH my_script.r
```

- In parallel:

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

## Single Program/Multiple Data (SPMD)

- SPMD is a programming *paradigm*.
- Not to be confused with SIMD.
- SPMD utilizes MIMD architecture computers.
- Arguably the simplest extension of serial programming.
- Difficult to describe, easy to do. . .
- Only one program is written, executed in batch on all processors.
- Different processors are autonomous; there is no manager.
- The dominant programming model for large machines.



Introduction	pbdr	pbdrMPI	GBD	Break	Stats eg's	pbdrDMAT	pbdrDMAT eg's	Wrapup
ooooooooooooo ooooooooooooo ooooo	oooo oooo	oooo oooooooooooo oooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo oooooooooo	ooo oooo oo	

# Contents

- 3 Introduction to pbdrMPI
  - Managing a Communicator
  - Reduce, Gather, Broadcast, and Barrier
  - Other pbdrMPI Tools

## 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

```

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

```



## 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:

```

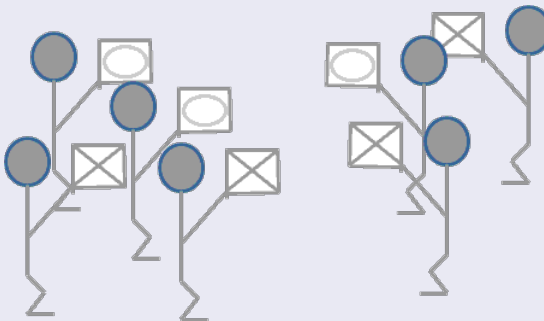
1 COMM.RANK = 0
2 [1] "Hello, world"
3 [1] "Hello again"
4 [1] "Hello again"

```

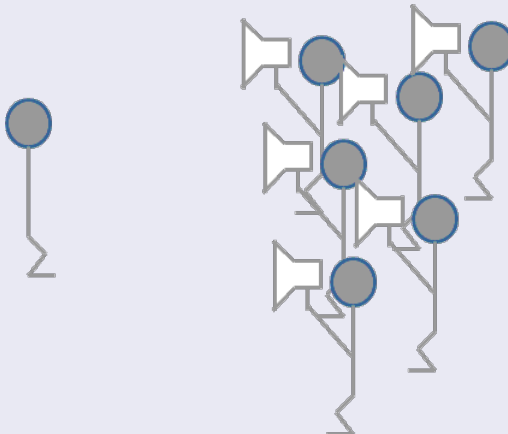
## MPI Operations

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

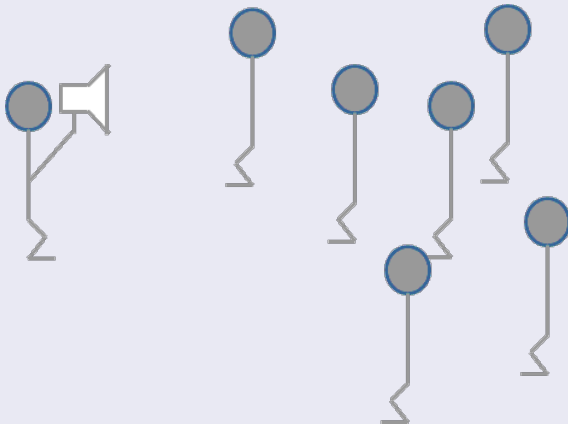
## Reductions — Combine results into single result



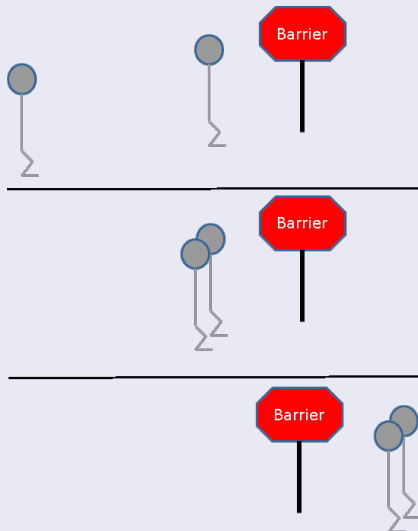
## Gather — Many-to-one



## Broadcast — One-to-many



## Barrier — Synchronization



## 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()`

## Quick Example 3

### Reduce and Gather: 3\_gt.r

```

1 library(pbdMPI, quiet = TRUE)
2 init()
3
4 comm.set.seed(diff=TRUE)
5
6 n <- sample(1:10, size=1)
7
8 gt <- gather(n)
9 comm.print(unlist(gt))
10
11 sm <- allreduce(n, op='sum')
12 comm.print(sm, all.rank=T)
13
14 finalize()

```

Execute this script via:

```
1 mpirun -np 2 Rscript 3_gt.r
```

Sample Output:

```

1 COMM.RANK = 0
2 [1] 2 8
3 COMM.RANK = 0
4 [1] 10
5 COMM.RANK = 1
6 [1] 10

```



## Quick Example 4

### Broadcast: 4\_bcast.r

```

1 library(pbdMPI, quiet=T)
2 init()
3
4 if (comm.rank()==0){
5   x <- matrix(1:4, nrow=2)
6 } else {
7   x <- NULL
8 }
9
10 y <- bcast(x, rank.source=0)
11
12 comm.print(y, rank=1)
13
14 finalize()

```

Execute this script via:

```
1 mpirun -np 2 Rscript 4_bcast.r
```

Sample Output:

```

1 COMM.RANK = 1
2   [,1] [,2]
3 [1,]   1   3
4 [2,]   2   4

```

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
ooooooooooooo ooooooooooooo ooooooooooooo ooooo	oooo oooo oooo	oooo ooooooooo ●ooooo	ooo ooo ooo		oooo ooo ooo	ooooo ooooooooo ooooooooooo	ooo oooo oo	

## 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>

## Quick Example 5

Barrier: 5\_barrier.r

```

1 library(pbdMPI, quiet = TRUE)
2 init()
3
4 .SPMD.CT$msg.bARRIER <- TRUE
5 .SPMD.CT$print.quiet <- TRUE
6
7 for (rank in 1:comm.size()-1){
8   if (comm.rank() == rank){
9     cat(paste("Hello", rank+1, "of", comm.size(), "\n"))
10  }
11  barrier()
12 }
13
14 comm.cat("\n")
15
16 comm.cat(paste("Hello", comm.rank()+1, "of",
17               comm.size(), "\n"), all.rank=TRUE)
18 finalize()

```

Execute this script via:

```
1 mpirun -np 2 Rscript 5_barrier.r
```

Sample Output:

```

1 Hello 1 of 2
2 Hello 2 of 2

```

## 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).

## Quick Example 6

Timing: 6\_timer.r

```

1 library(pbdMPI, quiet=TRUE)
2 init()
3
4 comm.set.seed(diff=T)
5
6 test <- function(timed)
7 {
8   ltime <- system.time(timed)[3]
9
10  mintime <- allreduce(ltime, op='min')
11  maxtime <- allreduce(ltime, op='max')
12  meantime <- allreduce(ltime, op='sum')/comm.size()
13
14  return(data.frame(min=mintime, mean=meantime,
15                    max=maxtime))
16 }
17 times <- test(rnorm(1e6)) # ~7.6MiB of data
18 comm.print(times)
19
20 finalize()

```

Execute this script via:

```
1 mpirun -np 2 Rscript 6_timer.r
```

Sample Output:

```

1      min  mean  max
2 1 0.17 0.173 0.176

```

## Other Helper Tools

**pbdBMPi** 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.  
pbdBApply(X, MARGIN, FUN, ...) — analogue of apply()  
pbdBLapply(X, FUN, ...) — analogue of lapply()  
pbdBsapply(X, FUN, ...) — analogue of sapply()

## Quick Comments for Using pbdrMPI

- 1 Start by loading the package:

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

- 2 Always initialize before starting and finalize when finished:

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
ooooooooooooo ooooooooooooo ooooo	oooo oooo	oooo ooooooooo ooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

# Contents

## 4 The Generalized Block Distribution

- The GBD Data Structure
- GBD: Example 1
- GBD: Example 2



## Distributing a Matrix Across 4 Processors: Block Distribution

	Data	Processors
$X =$	$X_{1,1}$ $X_{1,2}$ $X_{1,3}$	0
	$X_{2,1}$ $X_{2,2}$ $X_{2,3}$	1
	$X_{3,1}$ $X_{3,2}$ $X_{3,3}$	2
	$X_{4,1}$ $X_{4,2}$ $X_{4,3}$	3
	$X_{5,1}$ $X_{5,2}$ $X_{5,3}$	
	$X_{6,1}$ $X_{6,2}$ $X_{6,3}$	
	$X_{7,1}$ $X_{7,2}$ $X_{7,3}$	
	$X_{8,1}$ $X_{8,2}$ $X_{8,3}$	
	$X_{9,1}$ $X_{9,2}$ $X_{9,3}$	
	$X_{10,1}$ $X_{10,2}$ $X_{10,3}$	

$10 \times 3$

## Distributing a Matrix Across 4 Processors: Local Load Balance

	Data	Processors
$X =$	$X_{1,1}$	0
	$X_{1,2}$	1
	$X_{1,3}$	2
	$X_{2,1}$	3
	$X_{2,2}$	
	$X_{2,3}$	
	$X_{3,1}$	
	$X_{3,2}$	
	$X_{3,3}$	
	$X_{4,1}$	
	$X_{4,2}$	
	$X_{4,3}$	
	$X_{5,1}$	
	$X_{5,2}$	
	$X_{5,3}$	
	$X_{6,1}$	
	$X_{6,2}$	
	$X_{6,3}$	
	$X_{7,1}$	
	$X_{7,2}$	
	$X_{7,3}$	
	$X_{8,1}$	
	$X_{8,2}$	
	$X_{8,3}$	
	$X_{9,1}$	
	$X_{9,2}$	
	$X_{9,3}$	
	$X_{10,1}$	
	$X_{10,2}$	
	$X_{10,3}$	

$10 \times 3$

## 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.
- 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.

X <sub>1,1</sub>	X <sub>1,2</sub>	X <sub>1,3</sub>
X <sub>2,1</sub>	X <sub>2,2</sub>	X <sub>2,3</sub>
X <sub>3,1</sub>	X <sub>3,2</sub>	X <sub>3,3</sub>
X <sub>4,1</sub>	X <sub>4,2</sub>	X <sub>4,3</sub>
X <sub>5,1</sub>	X <sub>5,2</sub>	X <sub>5,3</sub>
X <sub>6,1</sub>	X <sub>6,2</sub>	X <sub>6,3</sub>
X <sub>7,1</sub>	X <sub>7,2</sub>	X <sub>7,3</sub>
X <sub>8,1</sub>	X <sub>8,2</sub>	X <sub>8,3</sub>
X <sub>9,1</sub>	X <sub>9,2</sub>	X <sub>9,3</sub>
X <sub>10,1</sub>	X <sub>10,2</sub>	X <sub>10,3</sub>

## 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

## 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

[	X <sub>11</sub>	X <sub>12</sub>	X <sub>13</sub>	X <sub>14</sub>	X <sub>15</sub>	X <sub>16</sub>	X <sub>17</sub>	X <sub>18</sub>	X <sub>19</sub>	]	2×9
[	X <sub>21</sub>	X <sub>22</sub>	X <sub>23</sub>	X <sub>24</sub>	X <sub>25</sub>	X <sub>26</sub>	X <sub>27</sub>	X <sub>28</sub>	X <sub>29</sub>	]	2×9
[	X <sub>31</sub>	X <sub>32</sub>	X <sub>33</sub>	X <sub>34</sub>	X <sub>35</sub>	X <sub>36</sub>	X <sub>37</sub>	X <sub>38</sub>	X <sub>39</sub>	]	2×9
[	X <sub>41</sub>	X <sub>42</sub>	X <sub>43</sub>	X <sub>44</sub>	X <sub>45</sub>	X <sub>46</sub>	X <sub>47</sub>	X <sub>48</sub>	X <sub>49</sub>	]	2×9
[	X <sub>51</sub>	X <sub>52</sub>	X <sub>53</sub>	X <sub>54</sub>	X <sub>55</sub>	X <sub>56</sub>	X <sub>57</sub>	X <sub>58</sub>	X <sub>59</sub>	]	2×9
[	X <sub>61</sub>	X <sub>62</sub>	X <sub>63</sub>	X <sub>64</sub>	X <sub>65</sub>	X <sub>66</sub>	X <sub>67</sub>	X <sub>68</sub>	X <sub>69</sub>	]	2×9
[	X <sub>71</sub>	X <sub>72</sub>	X <sub>73</sub>	X <sub>74</sub>	X <sub>75</sub>	X <sub>76</sub>	X <sub>77</sub>	X <sub>78</sub>	X <sub>79</sub>	]	1×9
[	X <sub>81</sub>	X <sub>82</sub>	X <sub>83</sub>	X <sub>84</sub>	X <sub>85</sub>	X <sub>86</sub>	X <sub>87</sub>	X <sub>88</sub>	X <sub>89</sub>	]	1×9
[	X <sub>91</sub>	X <sub>92</sub>	X <sub>93</sub>	X <sub>94</sub>	X <sub>95</sub>	X <sub>96</sub>	X <sub>97</sub>	X <sub>98</sub>	X <sub>99</sub>	]	1×9

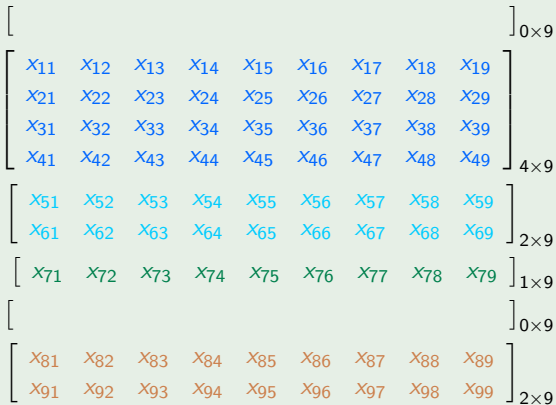
Processors = 0 1 2 3 4 5

## 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

## Understanding GBD: Local View



Processors = 0 1 2 3 4 5



Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo oooooooooooo ooooo	oooo oooo	oooo oooooooo ooooo	ooo ooo oo●		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

## 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.

# Brief Intermission

## Brief Intermission

### Questions? Comments?

Don't forget to talk to us at our discussion group:

<http://group.r-pbd.org/>

If you have an affiliation at a United States institution (university, research lab, etc.), consider getting an allocation with us:

<http://www.nics.tennessee.edu/getting-an-allocation>

Come to the talk *Elevating R to Supercomputers*, Friday, July 12th at 10:00 at the High Performance Computing session

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo	oooo	oooo	ooo		oooo	oooo	ooo	
oooooooooooo	oooo	oooooooo	ooo		ooo	oooo	oooo	
oooo		oooo	ooo		ooo	oooooooo	oo	

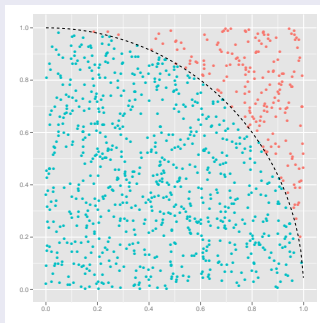
# Contents

- 6 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

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



## 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.

## 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()

```

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo	oooo	oooo	ooo		ooo●	oooo	ooo	
oooooooooooo	oooo	oooooooo	ooo		ooo	oooo	oooo	
oooo		oooo	ooo		ooo	oooooooo	oo	

## pbdMPI Example: Monte Carlo Simulation

### Note

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

## Example 2: Sample Covariance

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



## 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  $N - 1$ .

## 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)

```

### Example 3: Linear Regression

Find  $\beta$  such that

$$y = X\beta + \epsilon$$

When  $X$  is full rank,

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

### 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  $A^{-1} * B$

## Example 3: Linear Regression Code

### Serial Code

```

1 tX <- t(X)
2 A <- tX %*% X
3 B <- tX %*% y
4
5 ols <- solve(A) %*% B

```

### Parallel Code

```

1 tX.gbd <- t(X.gbd)
2 A <- allreduce(tX.gbd %*% X.gbd, op = "sum")
3 B <- allreduce(tX.gbd %*% y.gbd, op = "sum")
4
5 ols <- solve(A) %*% B

```

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo	oooo	oooo	ooo		oooo	oooo	ooo	
oooooooooooo	oooo	oooooooo	ooo		ooo	oooo	oooo	
ooooo		ooooo	ooo		ooo	oooooooo	oo	

# Contents

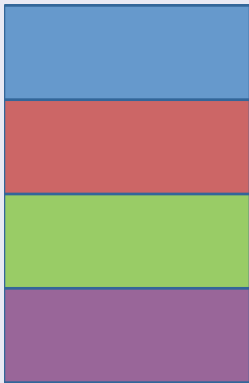
- 7 Introduction to pbdDMAT
  - Introduction to Distributed Matrices
  - DMAT Distributions
  - pbdDMAT

## Distributed Matrices

Most problems in data science are matrix algebra problems

- Data structure: block-cyclic matrix distributed across a 2-dimensional grid of processors.
- No single processor should hold all of the data.
- Very robust, but very confusing data structure.

## Distributed Matrices



(a) Block



(b) Cyclic

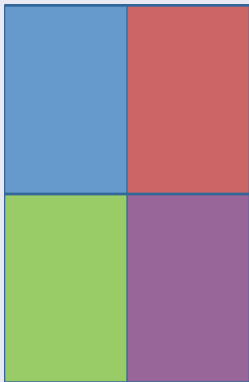


(c) Block-Cyclic

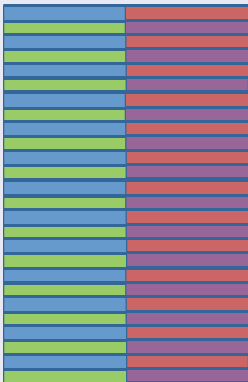
Figure: Matrix Distribution Schemes



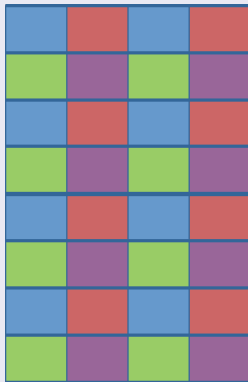
## Distributed Matrices



(a) 2d Block



(b) 2d Cyclic



(c) 2d Block-Cyclic

Figure: Matrix Distribution Schemes Onto a 2-Dimensional Grid

## Processor Grid Shapes

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

(a)  $1 \times 6$

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

(b)  $2 \times 3$

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

(c)  $3 \times 2$

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

(d)  $6 \times 1$

Table: Processor Grid Shapes with 6 Processors

## 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.

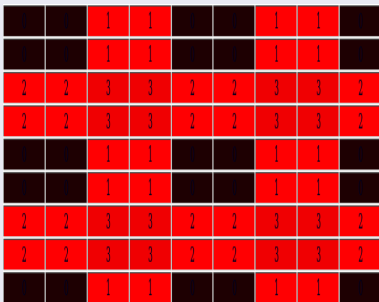
`ddmatrix` = {
 **Data**    S4 local submatrix, an R matrix  
**dim**     S4 dimension of the global matrix, a numeric pair  
**ldim**    S4 dimension of the local submatrix, a numeric pair  
**bldim**   S4 ScaLAPACK blocking factor, a numeric pair  
**CTXT**   S4 BLACS context, an numeric singleton

with prototype

`new("ddmatrix")` = {
 **Data**    = `matrix(0.0)`  
**dim**     = `c(1,1)`  
**ldim**    = `c(1,1)`  
**bldim**   = `c(1,1)`  
**CTXT**    = `0.0`

## Distributed Matrices: The Data Structure

Example: an  $9 \times 9$  matrix is distributed with a “block-cycling” factor of  $2 \times 2$  on a  $2 \times 2$  processor grid:



$$= \left\{ \begin{array}{ll} \text{Data} & = \text{matrix}(\dots) \\ \text{dim} & = \text{c}(9, 9) \\ \text{ldim} & = \text{c}(\dots) \\ \text{bldim} & = \text{c}(2, 2) \\ \text{CTXT} & = 0 \end{array} \right.$$

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

## 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

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

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

## DMAT: 2-dimensional Row Block

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

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

## DMAT: 1-dimensional Row Cyclic

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

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



## DMAT: 2-dimensional Row Cyclic

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

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

## DMAT: 2-dimensional Block-Cyclic

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

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
oooooooooooo oooooooooooo oooooooooooo ooooo	oooo oooo oooo	oooo oooooooo oooooo	ooo ooo ooo		oooo ooo ooo	oooo ooooo ●oooooo	ooo oooo oo	

## The DMAT Data Structure

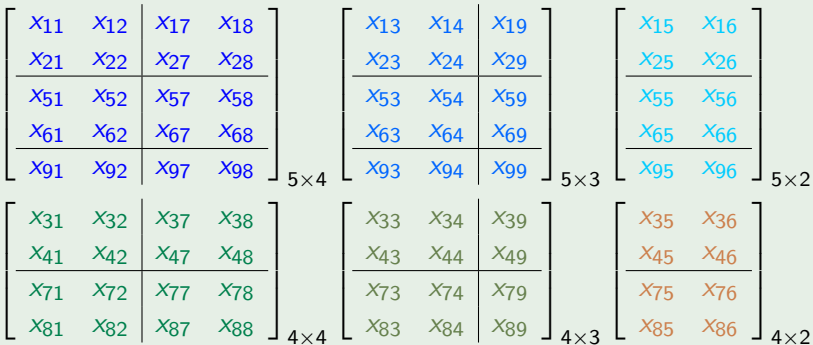
The more complicated the processor grid, the more complicated the distribution.

## DMAT: 2-dimensional Block-Cyclic with 6 Processors

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

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

## Understanding DMAT: Local View



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

## 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. . .
- 6 GBD is a generalization of the one-dimensional block DMAT distribution. Otherwise there is no relation.
- 7 DMAT is confusing, but very robust.

X <sub>11</sub>	X <sub>12</sub>	X <sub>13</sub>	X <sub>14</sub>	X <sub>15</sub>
X <sub>21</sub>	X <sub>22</sub>	X <sub>23</sub>	X <sub>24</sub>	X <sub>25</sub>
X <sub>31</sub>	X <sub>32</sub>	X <sub>33</sub>	X <sub>34</sub>	X <sub>35</sub>
X <sub>41</sub>	X <sub>42</sub>	X <sub>43</sub>	X <sub>44</sub>	X <sub>45</sub>
X <sub>51</sub>	X <sub>52</sub>	X <sub>53</sub>	X <sub>54</sub>	X <sub>55</sub>
X <sub>61</sub>	X <sub>62</sub>	X <sub>63</sub>	X <sub>64</sub>	X <sub>65</sub>
X <sub>71</sub>	X <sub>72</sub>	X <sub>73</sub>	X <sub>74</sub>	X <sub>75</sub>
X <sub>81</sub>	X <sub>82</sub>	X <sub>83</sub>	X <sub>84</sub>	X <sub>85</sub>
X <sub>91</sub>	X <sub>92</sub>	X <sub>93</sub>	X <sub>94</sub>	X <sub>95</sub>

## 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).

## 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** is MPI + some sugar.
- The GBD data structure is not the only thing **pbdMPI** can handle (just a useful convention).
- **pbdDMAT** is more of a software package.
- The block-cyclic DMAT structure *must* be used for **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.

## Sample Covariance

### Serial Code

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

### Parallel Code

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

## 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)

```

## Quick Example 3

### PCA: pca.r

```

1 library(pbdDMAT, quiet=T)
2 init.grid()
3
4 n <- 1e4
5 p <- 250
6
7 comm.set.seed(diff=T)
8 x.dmat <- ddmatrix("rnorm", nrow=n, ncol=p, mean=100, sd=25)
9
10 pca <- prcomp(x=x.dmat, retx=TRUE, scale=TRUE)
11 prop_var <- cumsum(pca$sdev)/sum(pca$sdev)
12 i <- max(min(which(prop_var > 0.9)) - 1, 1)
13
14 y.dmat <- pca$x[, 1:i]
15
16 comm.cat("\nCols: ", i, "\n", quiet=T)
17 comm.cat("%Cols: ", i/dim(x.dmat)[2], "\n\n", quiet=T)
18
19 finalize()

```

Execute this script via:

Sample Output:

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

```
1 Cols: 221
2 %Cols: 0.884
```

## Generating Random Data

Using randomly generated matrices is the best way to “get your feet wet” with the pbd tools. You can do this in 2 ways:

- 1 Generate a global matrix and distribute it.
- 2 Generate locally only what is needed.

## Example 1: Random Distributed Matrix Generation

Generate a global matrix and distribute it

```

1 library(pbdDMAT, quiet=TRUE)
2 init.grid()
3
4 # Common global on all processors --> distributed
5 comm.set.seed(diff=FALSE)
6 x <- matrix(rnorm(100), nrow=10, ncol=10)
7 x.dmat <- as.ddmatrix(x)
8
9 # Global on processor 0 --> distributed
10 if (comm.rank()==0){
11   x <- matrix(rnorm(100), nrow=10, ncol=10)
12 } else {
13   x <- NULL
14 }
15 x.dmat <- as.ddmatrix(x)
16
17 finalize()

```

## Example 2: Random Distributed Matrix Generation

Generate locally only what is needed

```

1 library(pbdDMAT, quiet=TRUE)
2 init.grid()
3
4 comm.set.seed(diff = TRUE) # good seeds via rlecuyer
5 x.dmat <- ddmatrix("rnorm", nrow=10, ncol=10)
6
7 finalize()

```



## Example 3: Random Distributed Matrix Generation

Generate locally only what is needed

```

1 library(pbdDMAT, quiet=TRUE)
2 init.grid()
3
4 zero.dmat <- ddmatrix(0, nrow=100, ncol=100)
5 id.dmat <- diag(1, nrow=100, ncol=100)
6
7 finalize()

```

## Example 4: Random Distributed Matrix Generation

### Convert between GBD and DMAT

```

1 library(pbdDEMO, quiet=TRUE)
2 init.grid()
3
4 comm.set.seed(diff = TRUE)
5
6 N.gbd <- 1 + comm.rank()
7 X.gbd <- matrix(rnorm(N.gbd * 3), ncol = 3)
8
9 # convert GBD to DMAT
10 X.dmat <- gbd2dmat(X.gbd)
11
12 # convert DMAT to GBD
13 new.X.gbd <- dmat2gbd(X.dmat)
14
15 # undistribute
16 X <- as.matrix(X.dmat)
17
18 finalize()

```

## Distributed Matrices

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	pbdDMAT	pbdDMAT eg's	Wrapup
ooooooooooooo ooooooooooooo ooooo	oooo oooo	oooo ooooooooo ooooo	ooo ooo ooo		oooo ooo ooo	ooooo oooooo ooooooooo	ooo oooo oo	

# Contents

## 9 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?

Don't forget to come to the talk:

*Elevating R to Supercomputers*

Friday, July 12th at 10:00

at the High Performance Computing session!

## Randomized SVD<sup>3</sup>

### 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  $Y = (AA^*)^q A\Omega$  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  $\ell$  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  $j = 1, 2, \dots, q$
- 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  $Y_j = Q_jR_j$ .
- 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 }
```

<sup>1</sup>Halko N, Martinsson P-G and Tropp J A 2011 Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions *SIAM Rev.* **53** 217–88

## Randomized 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

```

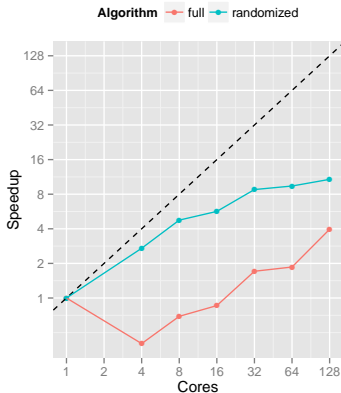
1 randSVD <- function(A, k, q=3)
2   {
3     ## Stage A
4     Omega <- ddmatrix("rnorm",
5                       nrow=n, ncol=2*k)
6     Y <- A %*% Omega
7     Q <- qr.Q(qr(Y))
8     At <- t(A)
9     for(i in 1:q)
10      {
11        Y <- At %*% Q
12        Q <- qr.Q(qr(Y))
13        Y <- A %*% Q
14        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   }

```



## Randomized SVD

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

