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[1], George Ostrouchov[1,2], Pragneshkumar Patel[2], Drew Schmidt[1]

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.

[1] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN
[2] 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	$1,000.5$	$1.000,5$

Contents

Contents

Quick Overview of Parallel Hardware

Three Basic Flavors of Hardware

Distributed Memory
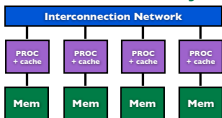
Shared Memory
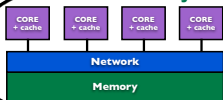
Co-Processor

GPU: Graphical Processing Unit
MIC: Many Integrated Core

Quick Overview of Parallel Hardware

Your Laptop or Desktop

Quick Overview of Parallel Hardware

A Server or Cluster

Distributed Memory

Interconnection Network

PROC + cache | PROC + cache | PROC + cache | PROC + cache

Mem | Mem | Mem | Mem

Co-Processor

GPU or MIC

Local Memory

GPU: Graphical Processing Unit
MIC: Many Integrated Core

Shared Memory

CORE + cache | CORE + cache | CORE + cache | CORE + cache

Network

Memory

Quick Overview of Parallel Hardware

Server to Supercomputer
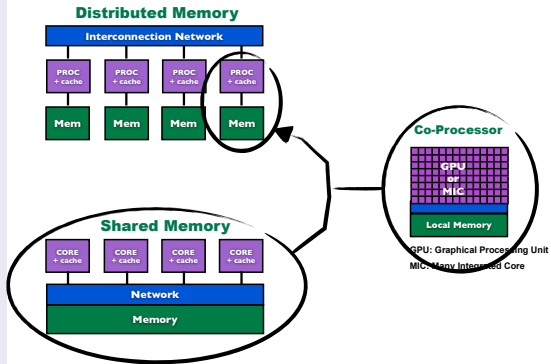
Knowing the Right Words
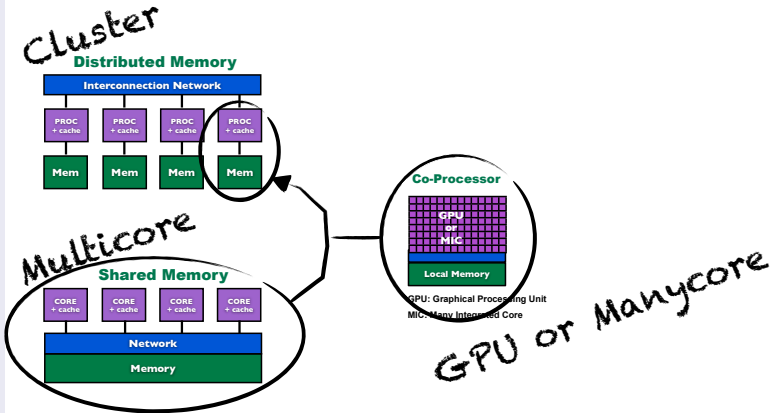
"Native" Programming Models and Tools

Distributed Memory

Interconnection Network

PROC + cache | PROC + cache | PROC + cache | PROC + cache

Mem | Mem | Mem | Mem

Focus on who owns what data and what communication is needed

MPI, Sockets

Co-Processor

GPU / MIC

Local Memory

Same Task on Blocks of data

CUDA, OpenCL

GPU: Graphical Processing Unit
MIC: Many Integrated Core

Shared Memory

CORE + cache | CORE + cache | CORE + cache | CORE + cache

Network

Memory

OpenMP

Focus on which tasks can be parallel

OpenMP, Threads, fork

R Interfaces to Native Tools
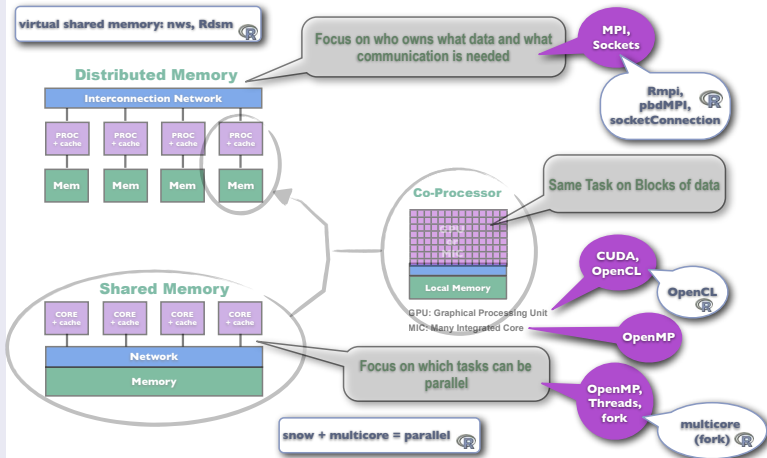
30+ Years of Parallel Computing Research
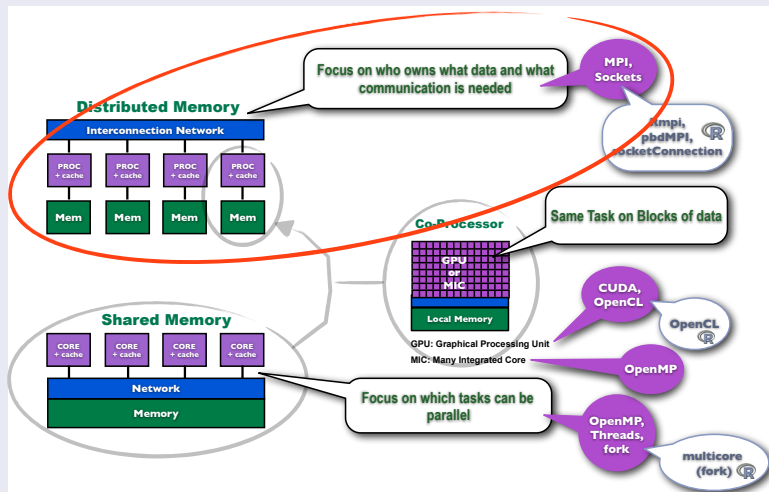
Last 10 years of Advances
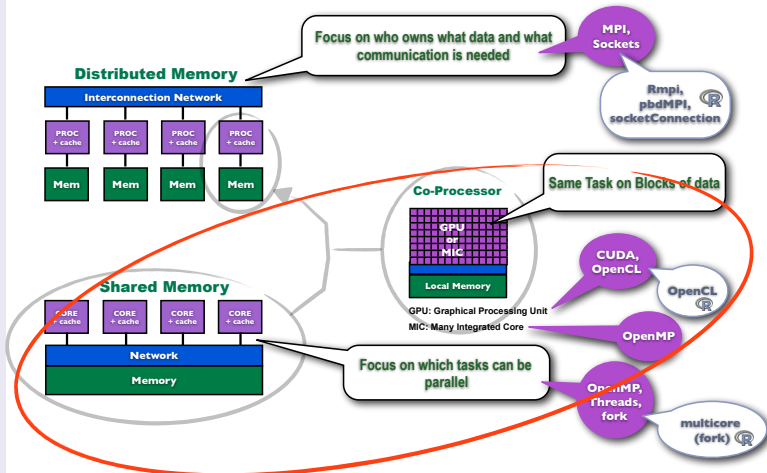
Putting It All Together Challenge
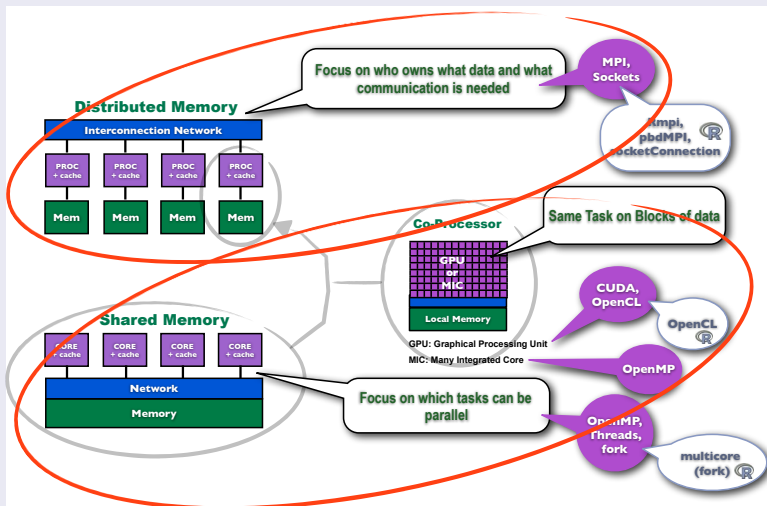
pbdR Focus on Data Parallelism
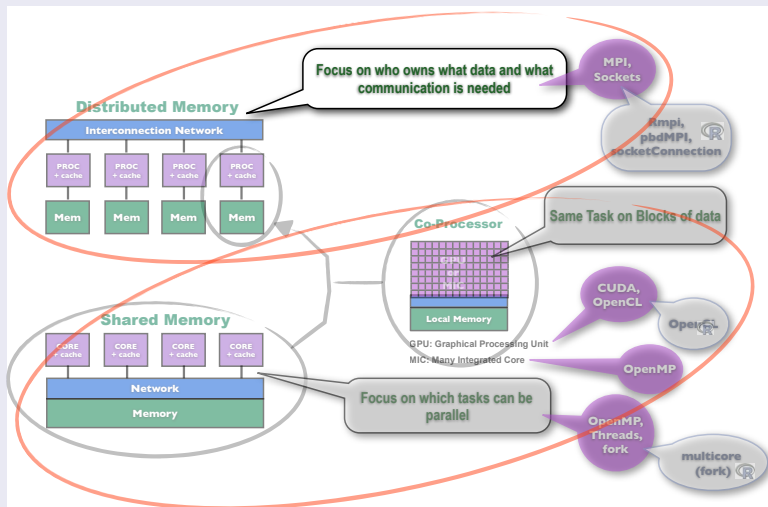
What is Parallelism?

- Doing more than one thing at a time.
- The simultaneous use of multiple compute resources to solve a computational problem.

Parallelism

Parallelism

Serial Programming
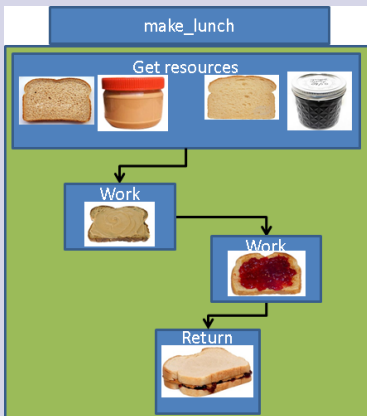
Parallel Programming
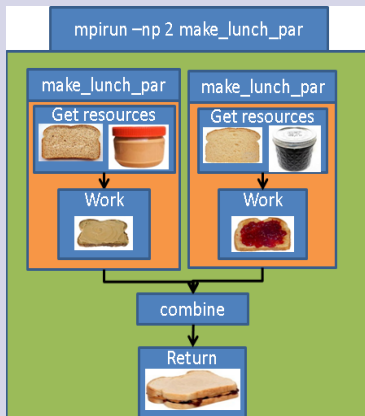
Kinds of Parallelism

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

(This is a gross oversimplification)

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

A Concise Introduction to Parallelism

pbdR Paradigms: Data Parallelism

Data parallelism:

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

Task parallelism:

- Often involves different tasks to the same data.

Data vs Task Parallelism
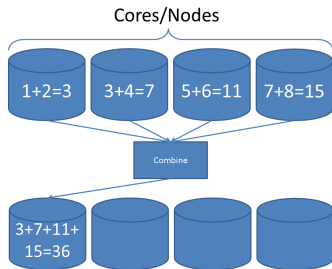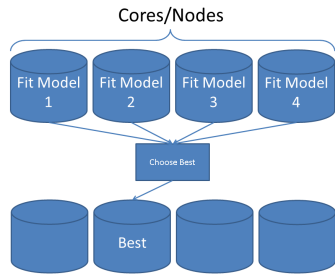
Parallel Programming Vocabulary: Difficulty in Parallelism

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.

A Concise Introduction to Parallelism

Speedup

- *Wallclock Time*: Time of the clock on the wall from start to finish

- *Speedup*: unitless measure of improvement; more is better.

$$S_{n_1,n_2} = \frac{\text{Run time for } n_1 \text{ cores}}{\text{Run time for } n_2 \text{ cores}}$$

  - n_1 is often taken to be 1
  - In this case, comparing parallel algorithm to serial algorithm

Speedup

Good Speedup
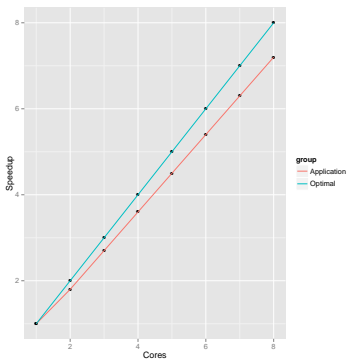
Bad Speedup
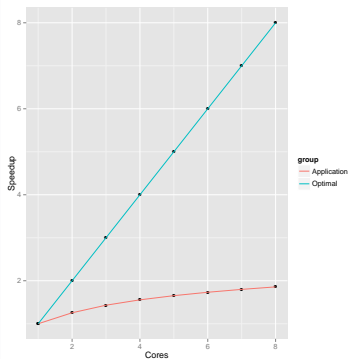
Recall: Shared and Distributed Memory Machines

Shared Memory

Direct access to read/change memory (one node)
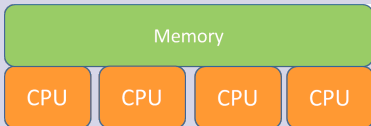
Distributed

No direct access to read/change memory (many nodes); requires communication
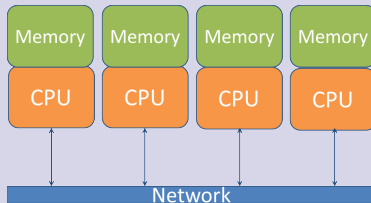
Shared and Distributed Memory Machines

Shared Memory Machines

Thousands of cores

Nautilus, University of Tennessee
1024 cores
4 TB RAM

Distributed Memory Machines

Hundreds of thousands of cores

Kraken, University of Tennessee
112,896 cores
147 TB RAM

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

R and Parallelism

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.

Why We Need Parallelism

1. Saves compute time.
2. Data size is skyrocketing.
3. Necessary for many problems.
4. Its necessity is coming.
5. *It's really cool.*

Recall: Parallel R Packages

Shared Memory

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

Distributed

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

(and others. . .)

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

R and Parallelism

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.

R and Parallelism

Likewise, the HPC community is looking for high-level languages for data...

Contents

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○○○○○○○○○	●○○○	○○○○	○○○○		○○○○	○○○○○○○○	○○○○	
○○○○○○○○○○○	○○○○○	○○○○○○○○	○○○		○○○	○○○○○○		
○○○○○○		○○○○○○	○○○		○○○	○○○○○○○	○○○	

The pbdR Project

Programming with Big Data in R (pbdR)

Striving for *Productivity, Portability, Performance*

Programming with Big Data in R

- *Free*[a] 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, ...)

---
[a]MPL, BSD, and GPL licensed

pbdR Packages
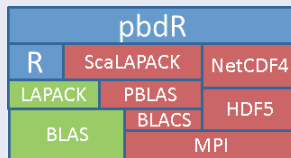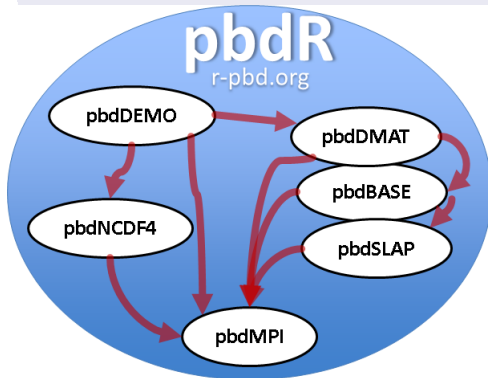
Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

The pbdR Project

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

Introduction   pbdR   pbdMPI   GBD   Break   Stats eg's   DMAT   pbdDMAT eg's   Wrapup
○○○○○○○○○○○○   ○○○○   ○○○○   ○○○○   ○○○○   ○○○○○○○○   ○○○○
○○○○○○○○○○○○   ○○○○○   ○○○○○○○○   ○○○   ○○○   ○○○○○○○   ○○○○
○○○○○○   ○○○○○○   ○○○   ○○○   ○○○○○○○○   ○○○

The pbdR Project

Least Squares Benchmark

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup
○○○○○○○○○○○○	○○○○	○○○○	○○○○		○○○○	○○○○○○○	○○○○	
○○○○○○○○○○○	●○○○○	○○○○○○○	○○○		○○○	○○○○○○	○○○○	
○○○○○○		○○○○○○	○○○		○○○	○○○○○○○	○○○	
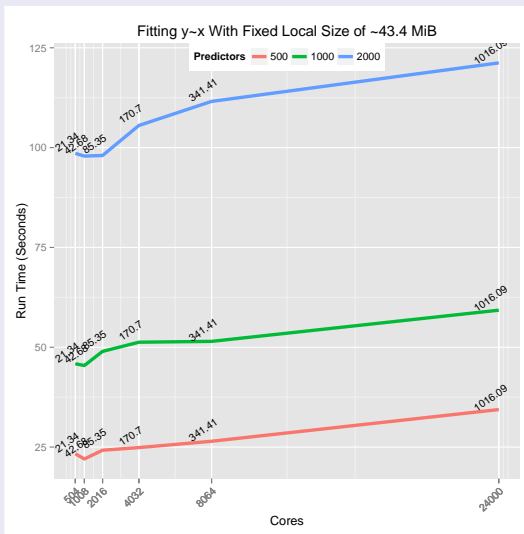
pbdR Paradigms

pbdR Paradigms

Programs that use pbdR utilize:

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

And generally utilize:

- Data Parallelism

Introduction	**pbdR**	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

pbdR Paradigms

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
```

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

pbdR Paradigms

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.

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

pbdR Paradigms

Single Program/Multiple Data (SPMD)

- 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.

SPMD

Manager/Worker

- Fascism
- Resentment
- Fealty

SPMD

- Democracy
- Cooperation
- Freeedom

Contents

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

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Managing a Communicator

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.

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Managing a Communicator

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
library(pbdMPI, quiet = TRUE)
init()

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

finalize()
```

Execute this script via:

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

Sample Output:

```
COMM.RANK = 0
[1] 0
COMM.RANK = 1
[1] 1
```

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Managing a Communicator

Quick Example 2

Hello World: 2_hello.r

```r
library(pbdMPI, quiet=TRUE)
init()

comm.print("Hello, world")

comm.print("Hello again", all.rank=TRUE, quiet=TRUE)

finalize()
```

Execute this script via:

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

Sample Output:

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

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Reduce, Gather, Broadcast, and Barrier

MPI Operations

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

Reductions — Combine results into single result

Gather — Many-to-one

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Reduce, Gather, Broadcast, and Barrier
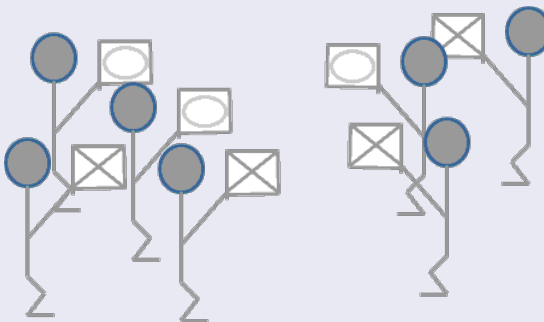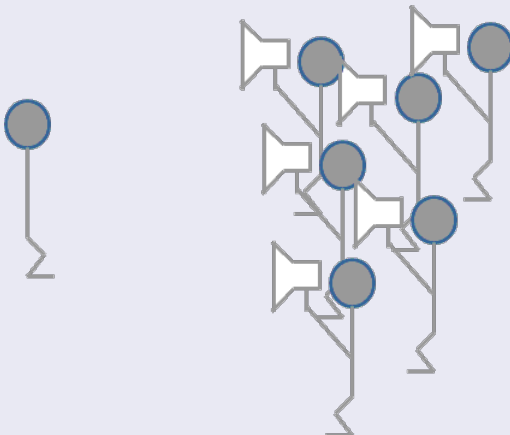
Broadcast — One-to-many
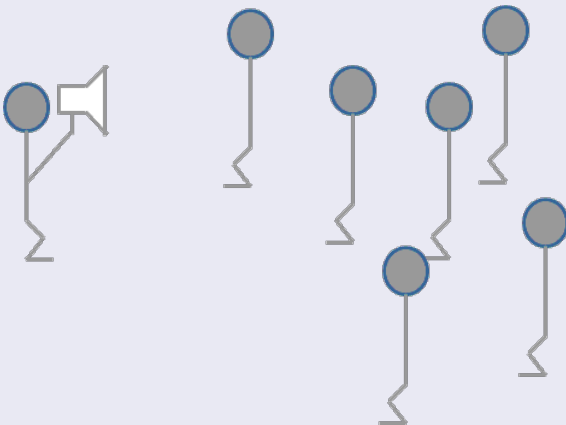
Barrier — Synchronization

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

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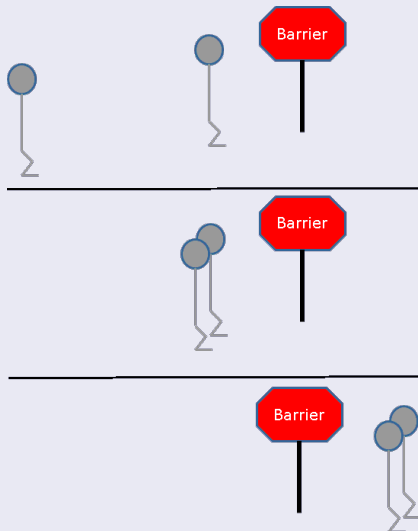
MPI Operations (2 of 2)

- **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()

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Reduce, Gather, Broadcast, and Barrier

Quick Example 3

Reduce and Gather: 3_gt.r

```r
library(pbdMPI, quiet = TRUE)
init()

comm.set.seed(diff=TRUE)

n <- sample(1:10, size=1)

gt <- gather(n)
comm.print(unlist(gt))

sm <- allreduce(n, op='sum')
comm.print(sm, all.rank=T)

finalize()
```

Execute this script via:

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

Sample Output:

```
COMM.RANK = 0
[1] 2 8
COMM.RANK = 0
[1] 10
COMM.RANK = 1
[1] 10
```

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Reduce, Gather, Broadcast, and Barrier

Quick Example 4

Broadcast: 4_bcast.r

```r
library(pbdMPI, quiet=T)
init()

if (comm.rank()==0){
  x <- matrix(1:4, nrow=2)
} else {
  x <- NULL
}

y <- bcast(x, rank.source=0)

comm.print(y, rank=1)

finalize()
```

Execute this script via:

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

Sample Output:

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

Introduction	pbdR	pbdMPI	GBD	Break	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

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Other pbdMPI Tools

Quick Example 5

Barrier: 5_barrier.r

```r
library(pbdMPI, quiet = TRUE)
init()

.SPMD.CT$msg.barrier <- TRUE
.SPMD.CT$print.quiet <- TRUE

for (rank in 1:comm.size()-1){
  if (comm.rank() == rank){
    cat(paste("Hello", rank+1, "of", comm.size(), "\n"))
  }
  barrier()
}

comm.cat("\n")

comm.cat(paste("Hello", comm.rank()+1, "of",
    comm.size(), "\n"), all.rank=TRUE)

finalize()
```

Execute this script via:

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

Sample Output:

```
Hello 1 of 2
Hello 2 of 2
```

Introduction	pbdR	pbdMPI	GBD	Break	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).

Introduction	pbdR	**pbdMPI**	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Other pbdMPI Tools

Quick Example 6

Timing: 6_timer.r

```r
library(pbdMPI, quiet=TRUE)
init()

comm.set.seed(diff=T)

test <- function(timed)
{
  ltime <- system.time(timed)[3]

  mintime <- allreduce(ltime, op='min')
  maxtime <- allreduce(ltime, op='max')
  meantime <- allreduce(ltime, op='sum')/comm.size()

  return(data.frame(min=mintime, mean=meantime,
      max=maxtime))
}

times <- test(rnorm(1e6)) # ~7.6MiB of data
comm.print(times)

finalize()
```

Execute this script via:

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

Sample Output:

```
    min   mean   max
1 0.17  0.173  0.176
```

Introduction	pbdR	**pbdMPI**	GBD	Break	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()`

Introduction	pbdR	pbdMPI	GBD	Break	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()
```

Contents

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

The GBD Data Structure

Distributing Data

Problem: How to distribute the data

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

?

Introduction	pbdR	pbdMPI	**GBD**	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

The GBD Data Structure

Distributing a Matrix Across 4 Processors: Block Distribution

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

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

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.

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

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

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

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

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

$$\begin{bmatrix} x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \end{bmatrix}_{1 \times 9}$$

$$\begin{bmatrix} x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99} \end{bmatrix}_{1 \times 9}$$

Processors = 0 1 2 3 4 5

Introduction	pbdR	pbdMPI	GBD	Break	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

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

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

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

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

pbdMPI Example: Monte Carlo Simulation
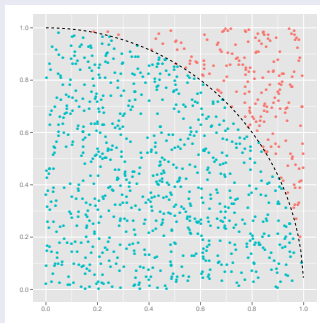
Example 1: Monte Carlo Simulation GBD Algorithm

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.

Introduction	pbdR	pbdMPI	GBD	Break	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()
```

Introduction	pbdR	pbdMPI	GBD	Break	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.

Example 2: Sample Covariance

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

Introduction	pbdR	pbdMPI	GBD	Break	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 $N - 1$.

Introduction	pbdR	pbdMPI	GBD	Break	**Stats eg's**	DMAT	pbdDMAT eg's	Wrapup

pbdMPI Example: Sample Covariance

Example 2: Sample Covariance Code

Serial Code

```r
N   <- nrow(X)
mu  <- colSums(X) / N

X   <- sweep(X, STATS=mu, MARGIN=2)
Cov.X <- crossprod(X) / (N-1)

print(Cov.X)
```

Parallel Code

```r
N   <- allreduce(nrow(X.gbd), op="sum")
mu  <- allreduce(colSums(X.gbd) / N, op="sum")

X.gbd <- sweep(X.gbd, STATS=mu, MARGIN=2)
Cov.X <- allreduce(crossprod(X.gbd), op="sum") / (N-1)

comm.print(Cov.X)
```

Introduction	pbdR	pbdMPI	GBD	Break	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,

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

Introduction	pbdR	pbdMPI	GBD	Break	**Stats eg's**	DMAT	pbdDMAT eg's	Wrapup

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 $A^{-1} * B$

Introduction   pbdR   pbdMPI   GBD   Break   **Stats eg's**   DMAT   pbdDMAT eg's   Wrapup
00000000000   0000   0000   0000   0000   00000000   0000
00000000000   00000   00000000   000   000   000000   0000
0000000   000000   000   00●   00000000   000

pbdMPI Example: Linear Regression

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
```

Contents

7 Introduction to pbdDMAT and the DMAT Structure
- Introduction to Distributed Matrices
- DMAT Distributions
- pbdDMAT

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Introduction to Distributed Matrices

Distributed Matrices

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

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

Introduction	pbdR	pbdMPI	GBD	Break	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. . .

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

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.

Introduction to Distributed Matrices

Distributed Matrices

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

Figure: Matrix Distribution Schemes

Introduction to Distributed Matrices

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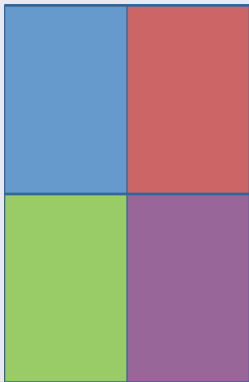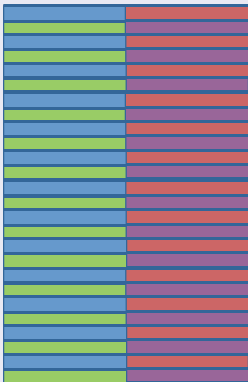
(a) 2d Block   (b) 2d Cyclic   (c) 2d Block-Cyclic
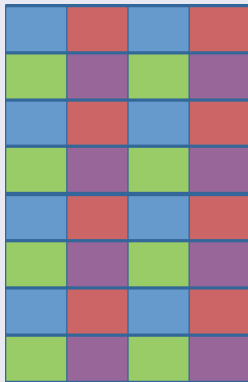
Figure: Matrix Distribution Schemes Onto a 2-Dimensional Grid

Processor Grid Shapes

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

(a) 1×6 (b) 2×3 (c) 3×2 (d) 6×1

Table: Processor Grid Shapes with 6 Processors

Introduction	pbdR	pbdMPI	GBD	Break	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.

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

with prototype

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

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:
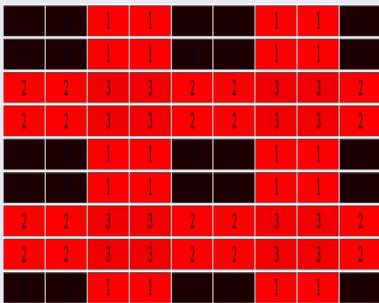
$$= \begin{cases} \textbf{Data} & = \texttt{matrix(...)} \\ \textbf{dim} & = \texttt{c(9, 9)} \\ \textbf{ldim} & = \texttt{c(...)} \\ \textbf{bldim} & = \texttt{c(2, 2)} \\ \textbf{CTXT} & = 0 \end{cases}$$

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

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

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

DMAT: 2-dimensional Row 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

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

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

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

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

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

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

pbdDMAT

The `DMAT` Data Structure

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

DMAT: 2-dimensional Block-Cyclic with 6 Processors

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

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

Introduction
pbdR
pbdMPI
GBD
Break
Stats eg's
DMAT
pbdDMAT eg's
Wrapup

pbdDMAT

Understanding DMAT: Local View

$$\begin{bmatrix} x_{11} & x_{12} & x_{17} & x_{18} \\ x_{21} & x_{22} & x_{27} & x_{28} \\ x_{51} & x_{52} & x_{57} & x_{58} \\ x_{61} & x_{62} & x_{67} & x_{68} \\ x_{91} & x_{92} & x_{97} & x_{98} \end{bmatrix}_{5\times4} \begin{bmatrix} x_{13} & x_{14} & x_{19} \\ x_{23} & x_{24} & x_{29} \\ x_{53} & x_{54} & x_{59} \\ x_{63} & x_{64} & x_{69} \\ x_{93} & x_{94} & x_{99} \end{bmatrix}_{5\times3} \begin{bmatrix} x_{15} & x_{16} \\ x_{25} & x_{26} \\ x_{55} & x_{56} \\ x_{65} & x_{66} \\ x_{95} & x_{96} \end{bmatrix}_{5\times2}$$

$$\begin{bmatrix} x_{31} & x_{32} & x_{37} & x_{38} \\ x_{41} & x_{42} & x_{47} & x_{48} \\ x_{71} & x_{72} & x_{77} & x_{78} \\ x_{81} & x_{82} & x_{87} & x_{88} \end{bmatrix}_{4\times4} \begin{bmatrix} x_{33} & x_{34} & x_{39} \\ x_{43} & x_{44} & x_{49} \\ x_{73} & x_{74} & x_{79} \\ x_{83} & x_{84} & x_{89} \end{bmatrix}_{4\times3} \begin{bmatrix} x_{35} & x_{36} \\ x_{45} & x_{46} \\ x_{75} & x_{76} \\ x_{85} & x_{86} \end{bmatrix}_{4\times2}$$

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

Introduction	pbdR	pbdMPI	GBD	Break	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.

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

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).

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

pbdDMAT

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)
```

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	**DMAT**	pbdDMAT eg's	Wrapup

pbdDMAT

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.

Introduction	pbdR	pbdMPI	GBD	Break	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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Manipulating DMAT Objects

Example 1: DMAT Construction

Generate a global matrix and distribute it

```r
library(pbdDMAT, quiet=TRUE)
init.grid()

# Common global on all processors --> distributed
x <- matrix(1:100, nrow=10, ncol=10)
x.dmat <- as.ddmatrix(x)

x.dmat

# Global on processor 0 --> distributed
if (comm.rank()==0){
  y <- matrix(1:100, nrow=10, ncol=10)
} else {
  y <- NULL
}
y.dmat <- as.ddmatrix(y)

y.dmat

finalize()
```

Execute this script via:

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Manipulating DMAT Objects

Example 2: DMAT Construction

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  zero.dmat
6
7  id.dmat <- diag(1, nrow=100, ncol=100, type="ddmatrix")
8  id.dmat
9
10 comm.set.seed(diff=T)
11 rand.dmat <- ddmatrix("rnorm", nrow=100, ncol=100,
        mean=10, sd=100)
12 rand.dmat
13
14 finalize()
```

Execute this script via:

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Manipulating DMAT Objects

Example 3: DMAT Operations

Generate locally only what is needed

```
1  library(pbdDMAT, quiet=TRUE)
2  init.grid()
3
4  x.dmat <- ddmatrix(1:30, nrow=10)
5  y.dmat <- x.dmat[c(1, 3, 5, 7, 9), -3]
6
7  comm.print(y.dmat)
8  y <- as.matrix(y.dmat)
9  comm.print(y)
10
11 finalize()
```

Execute this script via:

```
1  mpirun —np 2 Rscript 3_extract.r
```

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	**pbdDMAT eg's**	Wrapup

Manipulating DMAT Objects

Example 4: More DMAT Operations

```
1  library(pbdDMAT, quiet=TRUE)
2  init.grid()
3
4  x.dmat <- ddmatrix(1:30, nrow=10)
5
6  y.dmat <- x.dmat + 1:7
7
8  z.dmat <- scale(x.dmat, center=TRUE, scale=TRUE)
9
10 y <- as.matrix(y.dmat)
11 z <- as.matrix(z.dmat)
12
13 comm.print(y)
14 comm.print(z)
15
16 finalize()
```

Execute this script via:

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

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)
```

Introduction	pbdR	pbdMPI	GBD	Break	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)
```

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

Statistics Examples with pbdDMAT

Example 5: PCA

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:

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

Sample Output:

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

Statistics Examples with pbdDMAT

Distributed Matrices

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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

RandSVD

Randomized SVD[3]

Serial R

PROTOTYPE FOR RANDOMIZED SVD

Given an $m \times n$ matrix A, a target number k of singular vectors, and an exponent q (say, $q = 1$ or $q = 2$), this procedure computes an approximate rank-$2k$ factorization $U\Sigma V^*$, where U and V are orthonormal, and Σ is nonnegative and diagonal.

Stage A:
1. Generate an $n \times 2k$ Gaussian test matrix Ω.
2. Form $Y = (AA^*)^q A\Omega$ by multiplying alternately with A and A^*.
3. Construct a matrix Q whose columns form an orthonormal basis for the range of Y.

Stage B:
4. Form $B = Q^*A$.
5. Compute an SVD of the small matrix: $B = \tilde{U}\Sigma V^*$.
6. Set $U = Q\tilde{U}$.

Note: The computation of Y in step 2 is vulnerable to round-off errors. When high accuracy is required, we must incorporate an orthonormalization step between each application of A and A^*; see Algorithm 4.4.

ALGORITHM 4.4: RANDOMIZED SUBSPACE ITERATION

Given an $m \times n$ matrix A and integers ℓ and q, this algorithm computes an $m \times \ell$ orthonormal matrix Q whose range approximates the range of A.
1. Draw an $n \times \ell$ standard Gaussian matrix Ω.
2. Form $Y_0 = A\Omega$ and compute its QR factorization $Y_0 = Q_0 R_0$.
3. **for** $j = 1, 2, \ldots, q$
4.   Form $\tilde{Y}_j = A^*Q_{j-1}$ and compute its QR factorization $\tilde{Y}_j = \tilde{Q}_j \tilde{R}_j$.
5.   Form $Y_j = A\tilde{Q}_j$ and compute its QR factorization $Y_j = Q_j R_j$.
6. **end**
7. $Q = Q_q$.

```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    }
```

[1] 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

Introduction	pbdR	pbdMPI	GBD	Break	Stats eg's	DMAT	pbdDMAT eg's	Wrapup

RandSVD

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  }
```
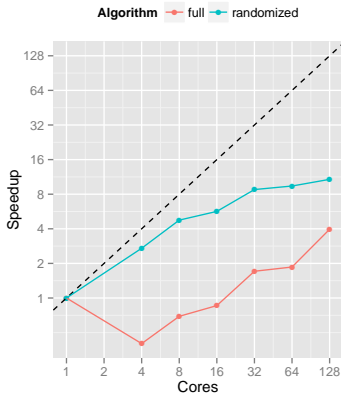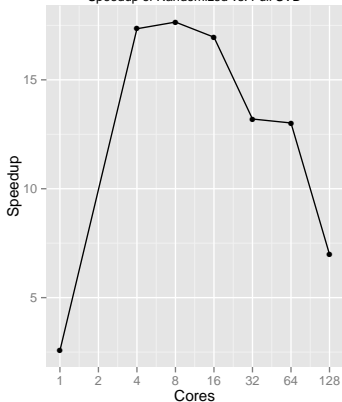
Parallel pbdR

```
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  }
```

RandSVD

Randomized SVD

Introduction
○○○○○○○○○○○○
○○○○○○○○○○○○
○○○○○○

pbdR
○○○○
○○○○○

pbdMPI
○○○○
○○○○○○○○○
○○○○○○

GBD
○○○○
○○○
○○○

Break

Stats eg's
○○○○
○○○
○○○

DMAT
○○○○○○○○
○○○○○○
○○○○○○○○

pbdDMAT eg's
○○○○
○○○○
○○○

Wrapup

Contents

1. **Wrapup** (9)

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`

Tutorials

- OLCF Data Workshop, August 8, Oak Ridge National Laboratory
- SC13, November 17-22, Denver, Colorado

Invited Talks

- JSM 2013, August 3-8, Montréal, Québec
- IASC, Aug 22-23, Seoul
- World Statistics Congress, August 25-30, Hong Kong

Thanks for coming!

Questions? Comments?