

pbdR: Programming with Big Data in R

pbdR Core Team

Tokyo, February 17-18, 2014



The **pbdR** Core Team

Wei-Chen Chen¹

George Ostrouchov^{2,3}

Pragneshkumar Patel³

Drew Schmidt³



Support

This work used resources of [National Institute for Computational Sciences](#) at the University of Tennessee, Knoxville, which is supported by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICCS-RDAV center. This work also used resources of the [Oak Ridge Leadership Computing Facility](#) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

¹Department of Ecology and Evolutionary Biology
University of Tennessee, Knoxville TN, USA

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

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

About This Presentation

Downloads

This presentation is available at:

[https://github.com/wrathematics/pbd-tutorial/blob/
master/pbdr/useR/pbdr.pdf?raw=true](https://github.com/wrathematics/pbd-tutorial/blob/master/pbdr/useR/pbdr.pdf?raw=true)

Scripts for exercises are at:

[https://github.com/wrathematics/pbd-tutorial/tree/
master/pbdr/common/scripts](https://github.com/wrathematics/pbd-tutorial/tree/master/pbdr/common/scripts)

About This Presentation

Speaking Serial R with a Parallel Accent

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

<http://goo.gl/HZkRt>

It contains more examples, and sometimes added detail.



About This Presentation

Installation Instructions

Installation instructions for setting up a pbdR environment are available:

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

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





Contents

- 1 Introduction
- 2 pbdR
- 3 Introduction to pbdMPI
- 4 The Generalized Block Distribution
- 5 Basic Statistics Examples
- 6 Data Input
- 7 Introduction to pbdDMAT and the DMAT Structure
- 8 Examples Using pbdDMAT
- 9 Profiling
- 10 Example Applications
- 11 Wrapup

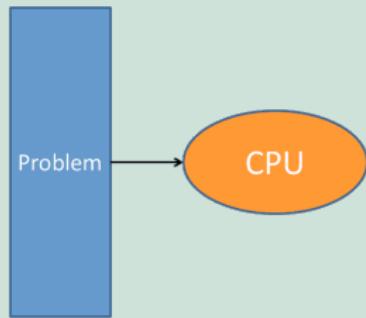
Contents

1 Introduction

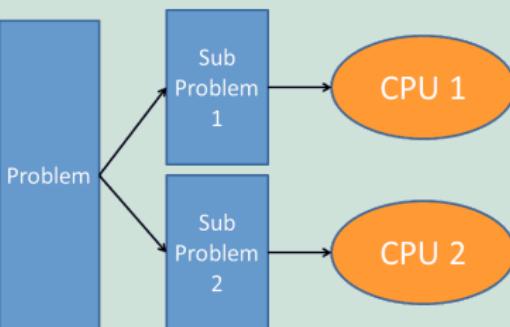
- A Concise Introduction to Parallelism
- Quick Overview of Parallel Hardware
- A Quick Overview of Parallel Software

Parallelism

Serial Programming



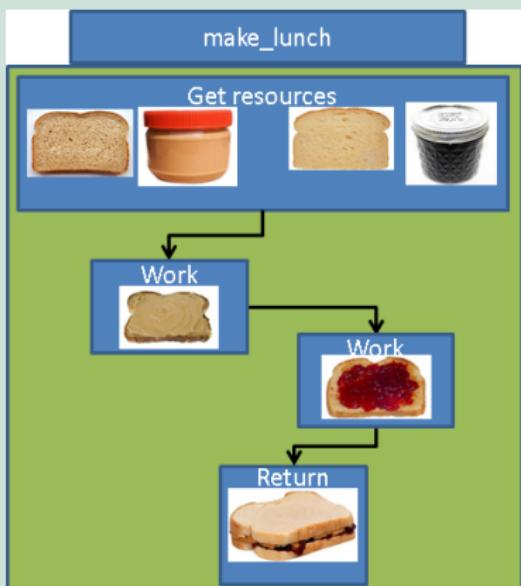
Parallel Programming



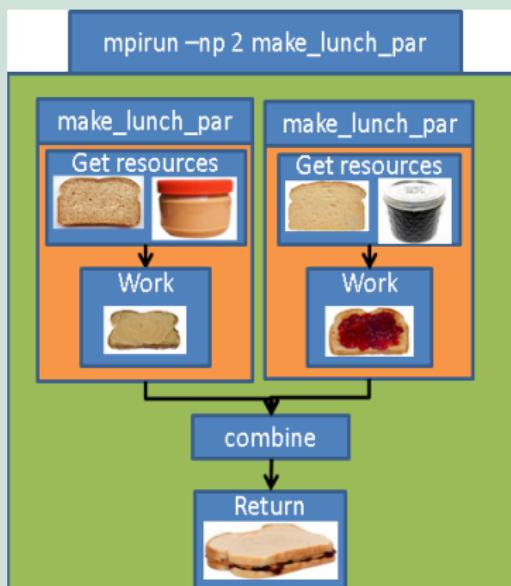
A Concise Introduction to Parallelism

Parallelism

Serial Programming



Parallel Programming





A Concise Introduction to 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.

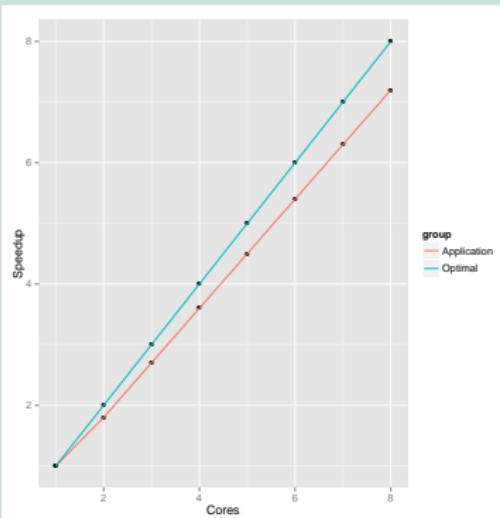
$$S_{n_1, n_2} = \frac{\text{Run time for } n_1 \text{ cores}}{\text{Run time for } n_2 \text{ cores}}$$

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

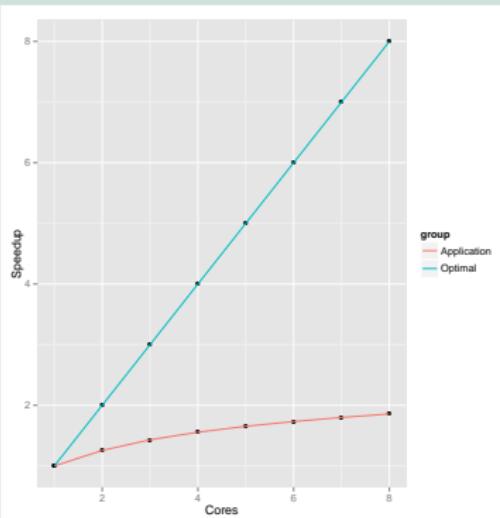
A Concise Introduction to Parallelism

Speedup

Good Speedup



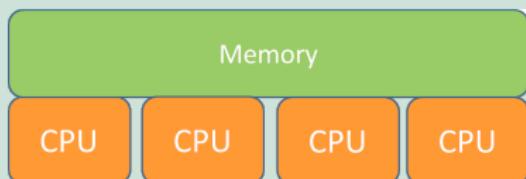
Bad Speedup



Shared and Distributed Memory Machines

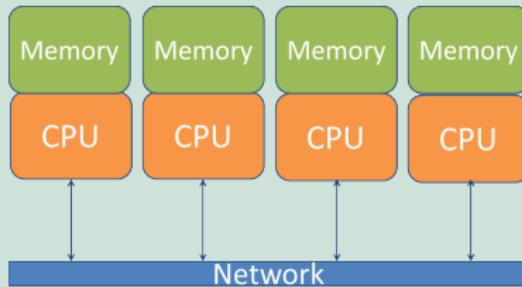
Shared Memory

Direct access to read/change memory (one node)



Distributed

No direct access to
read/change memory (many
nodes); requires communication





A Concise Introduction to Parallelism

Shared and Distributed Memory Machines

Shared Memory Machines

Thousands of cores



Nautilus, University of Tennessee
1024 cores
4 TB RAM

Distributed Memory Machines

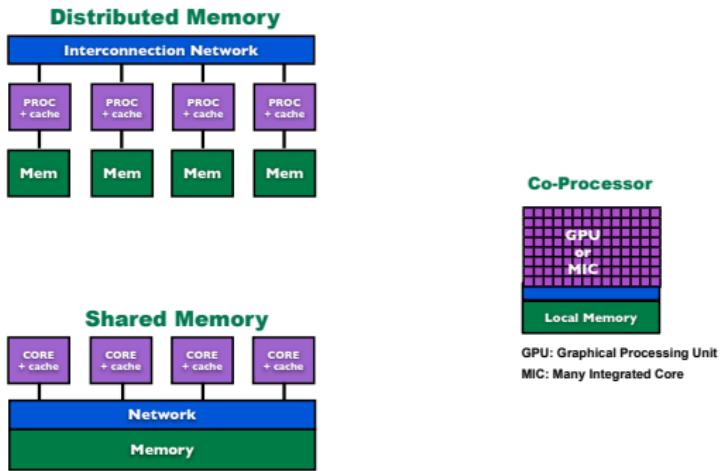
Hundreds of thousands of cores



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

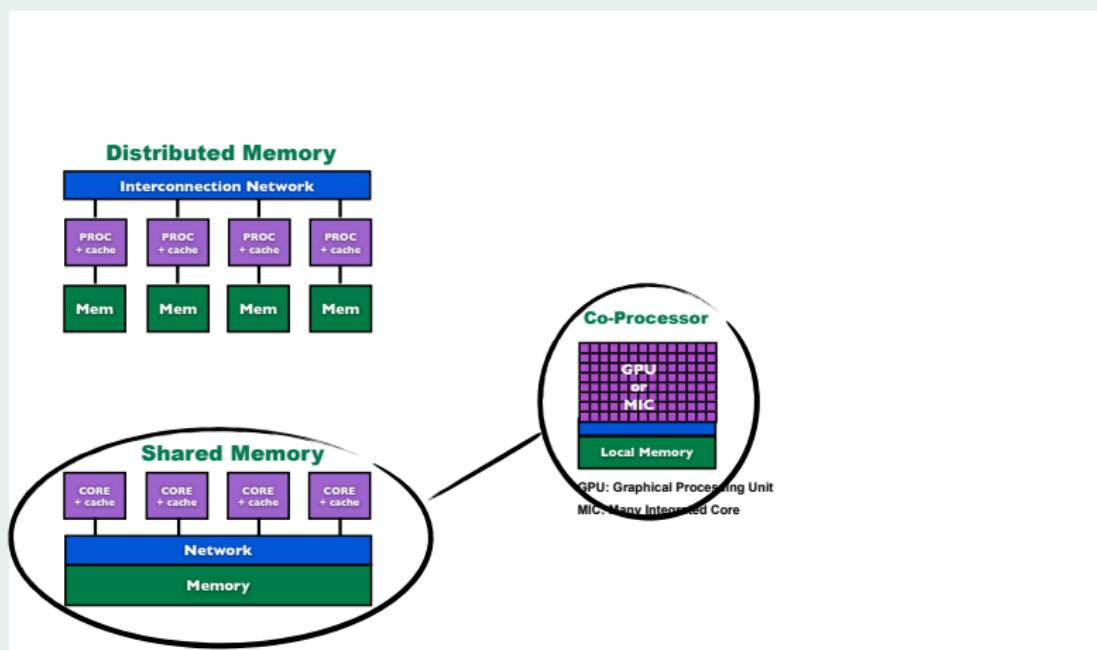
Quick Overview of Parallel Hardware

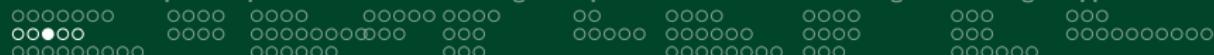
Three Basic Flavors of Hardware



Quick Overview of Parallel Hardware

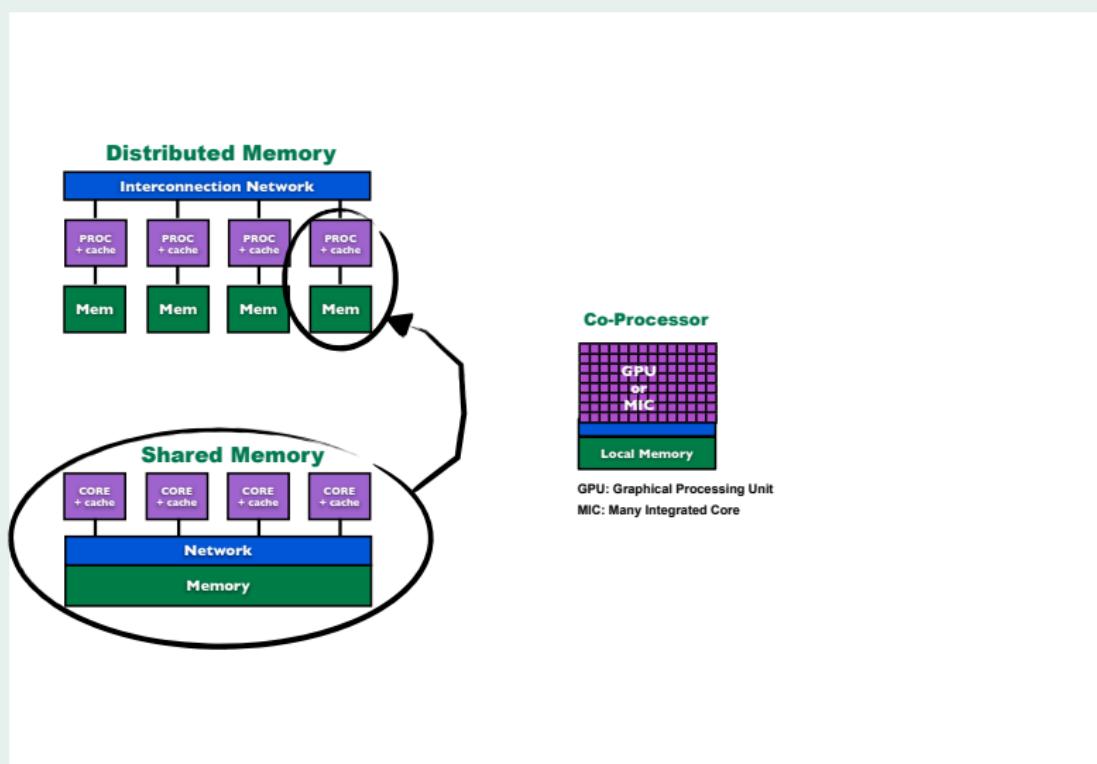
Your Laptop or Desktop

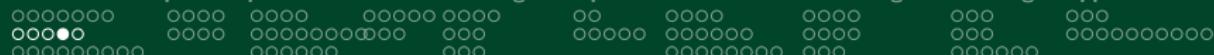




Quick Overview of Parallel Hardware

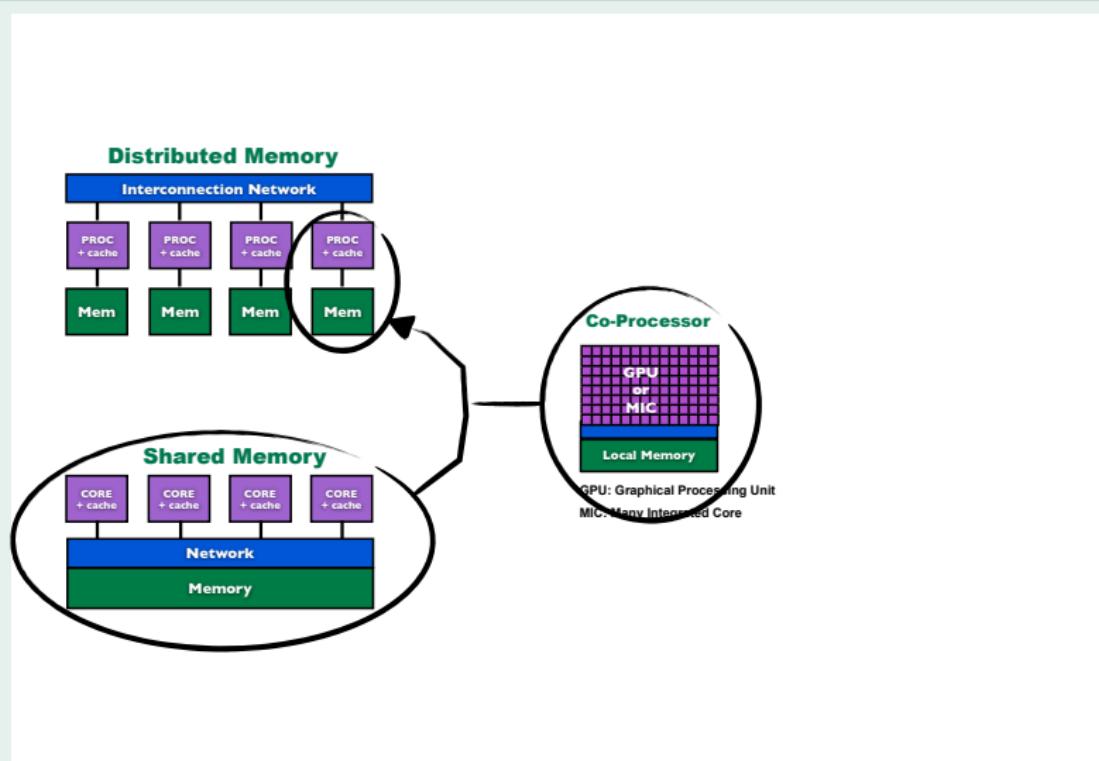
A Server or Cluster

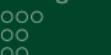




Quick Overview of Parallel Hardware

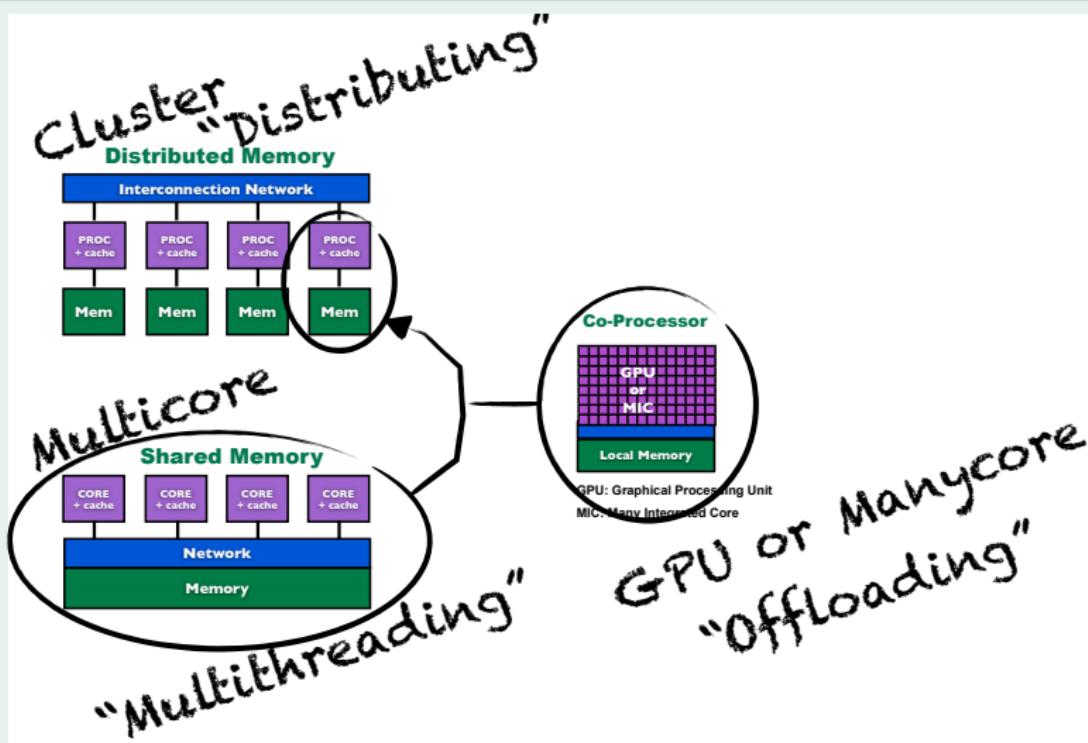
Server to Supercomputer

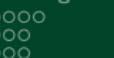




Quick Overview of Parallel Hardware

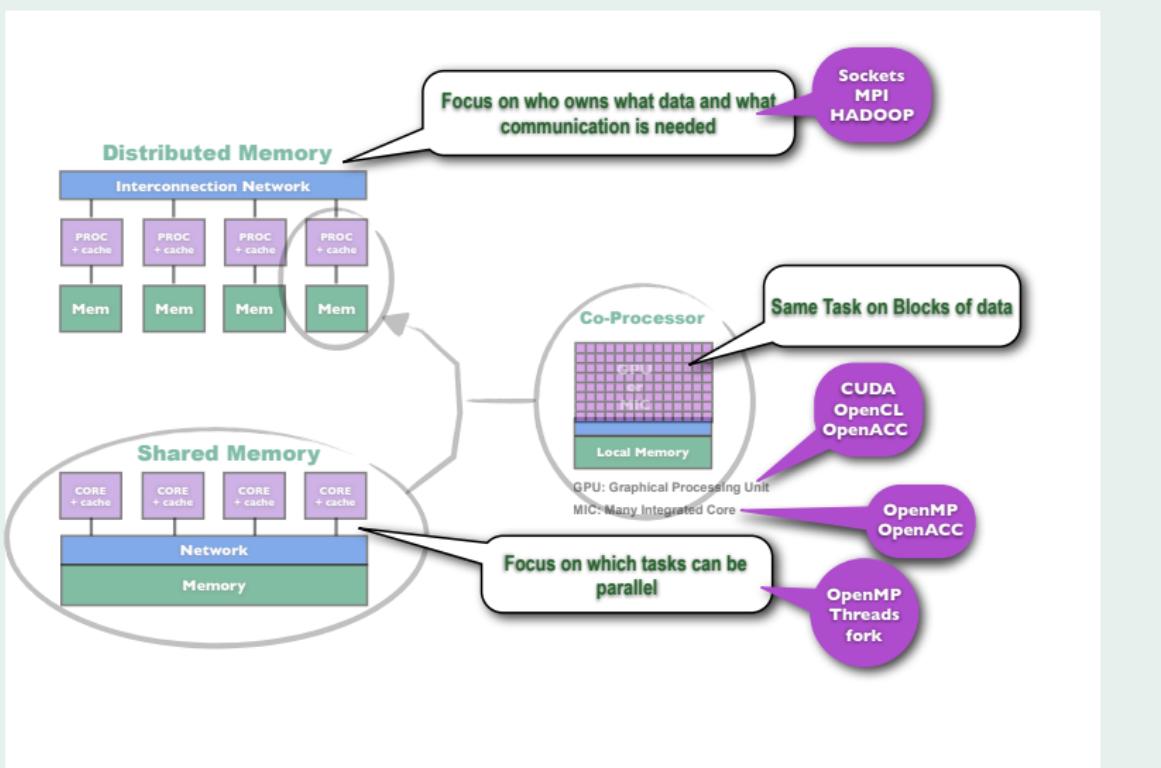
Knowing the Right Words





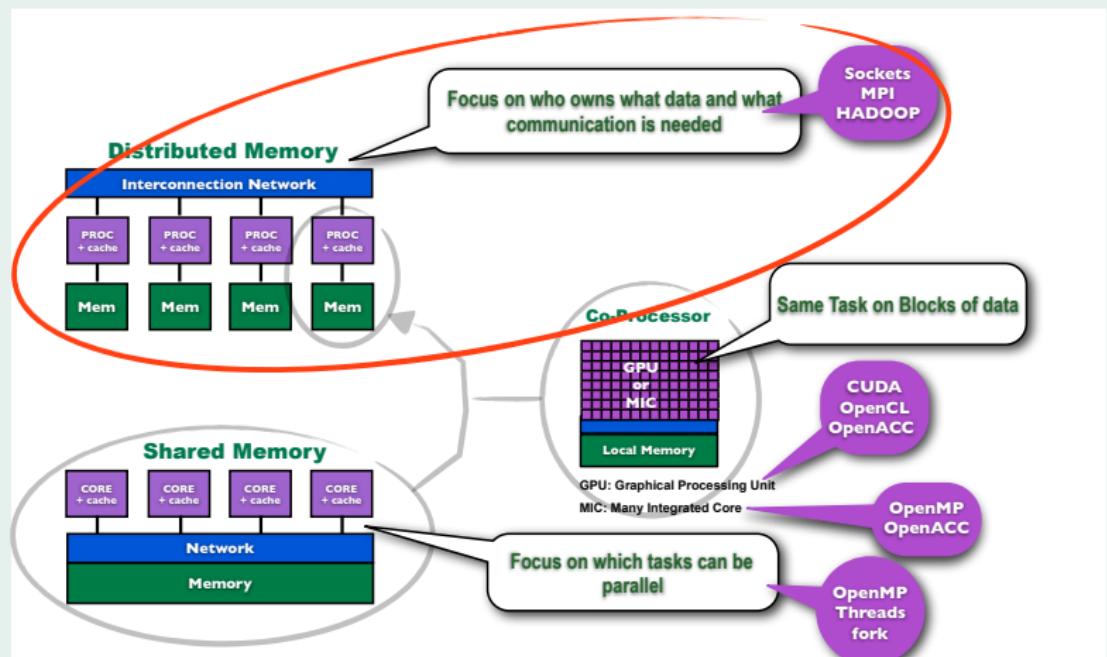
A Quick Overview of Parallel Software

“Native” Programming Models and Tools



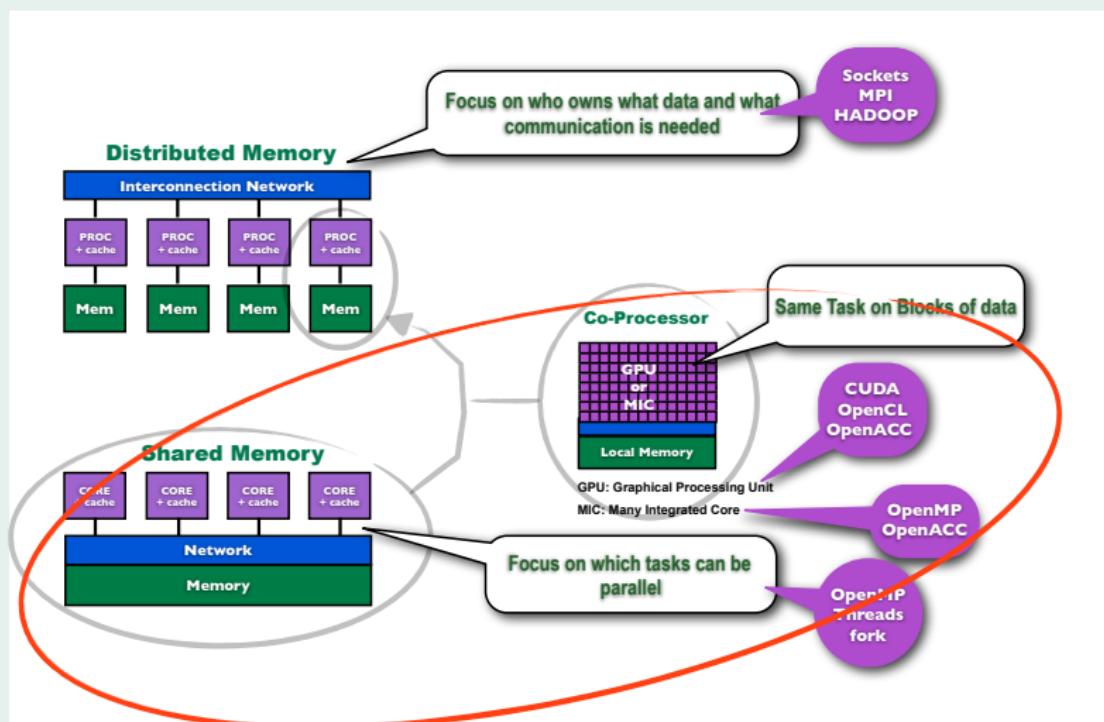
A Quick Overview of Parallel Software

30+ Years of Parallel Computing Research



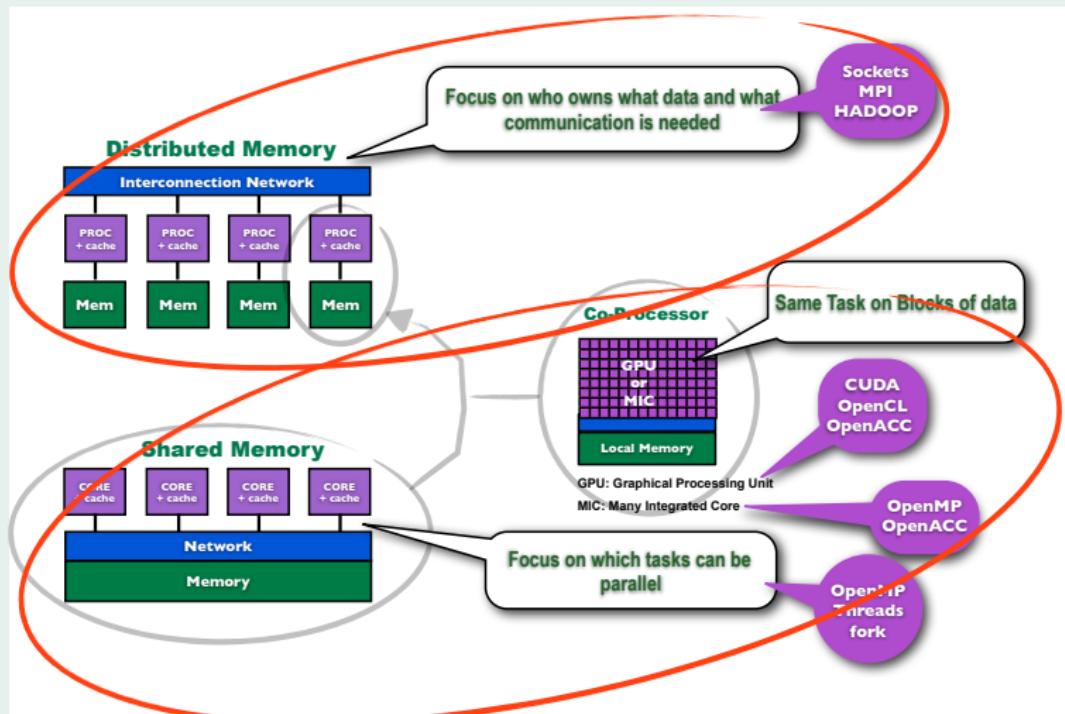
A Quick Overview of Parallel Software

Last 10 years of Advances



A Quick Overview of Parallel Software

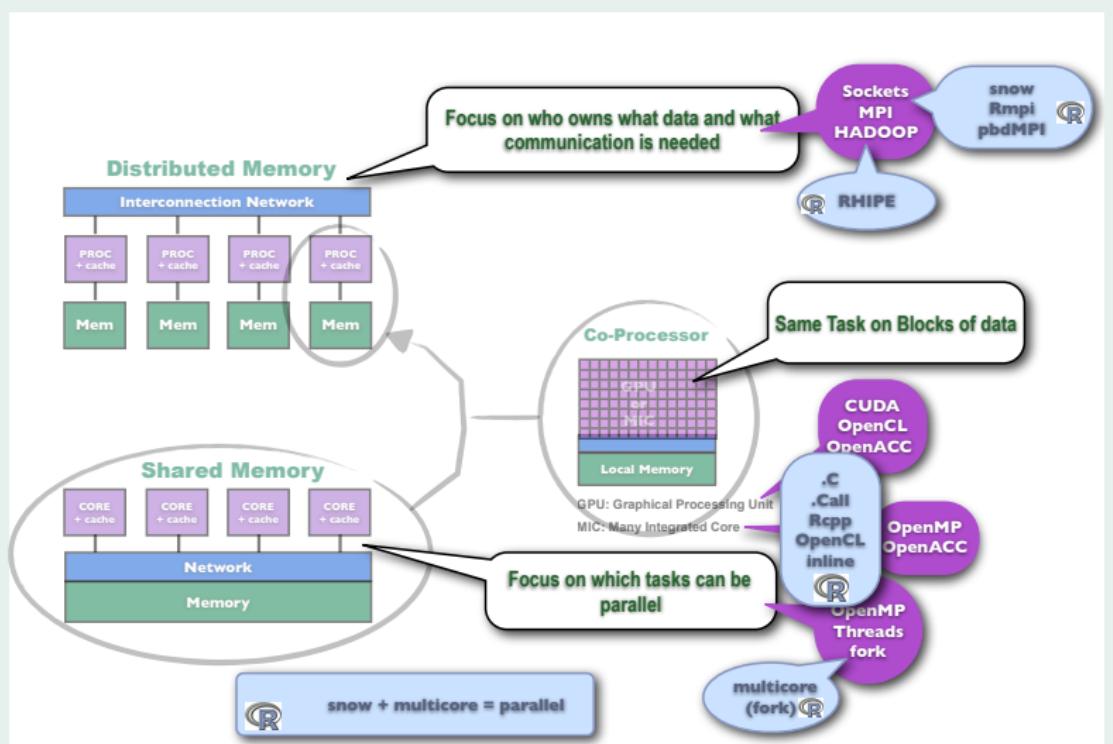
Putting It All Together Challenge

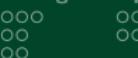




A Quick Overview of Parallel Software

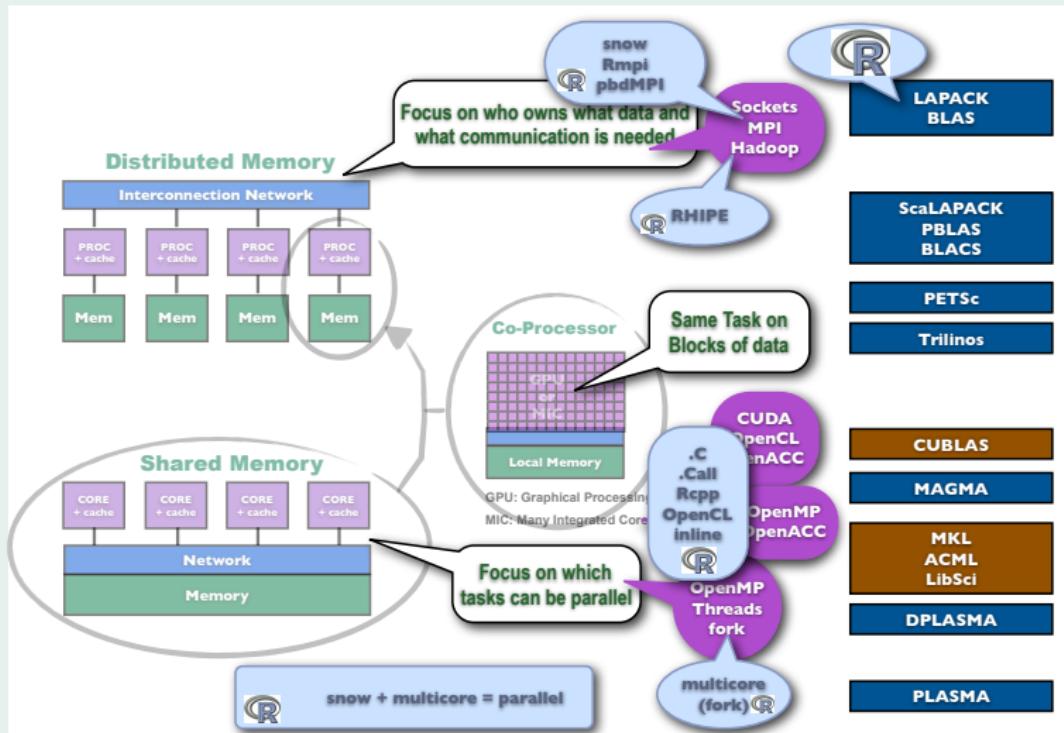
R Interfaces to Native Tools





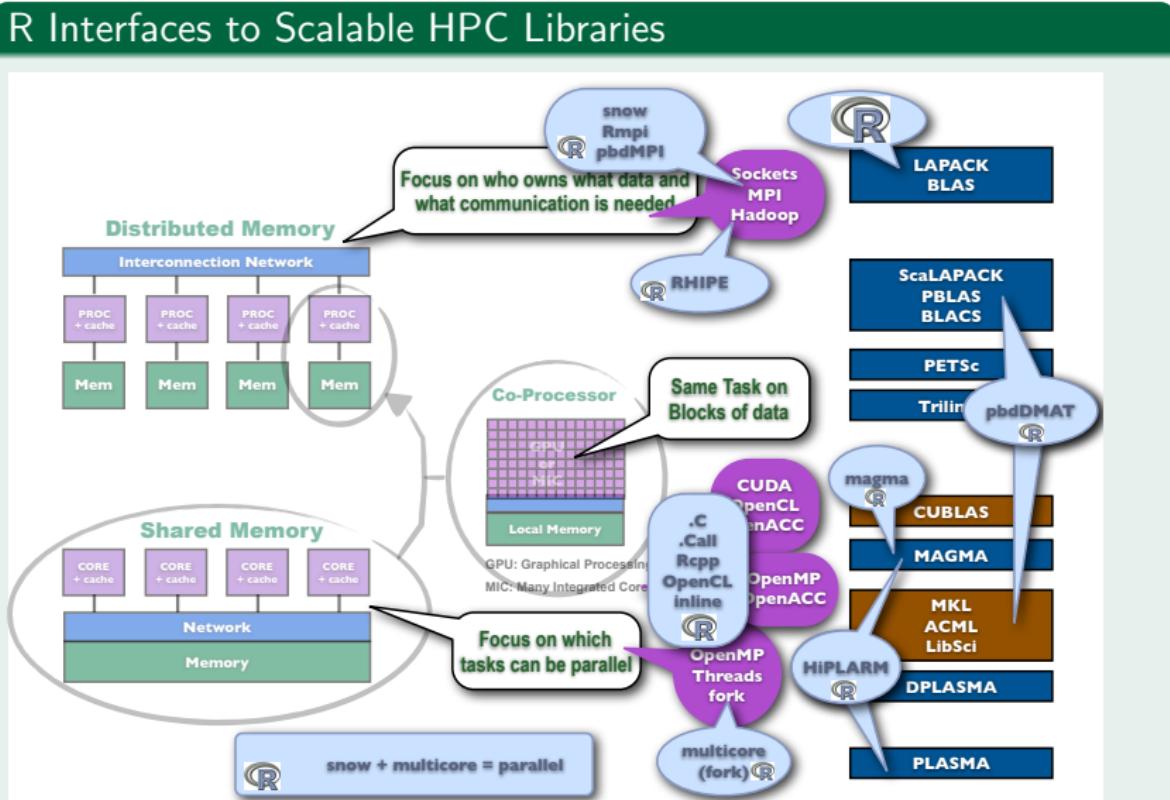
A Quick Overview of Parallel Software

HPC Libraries



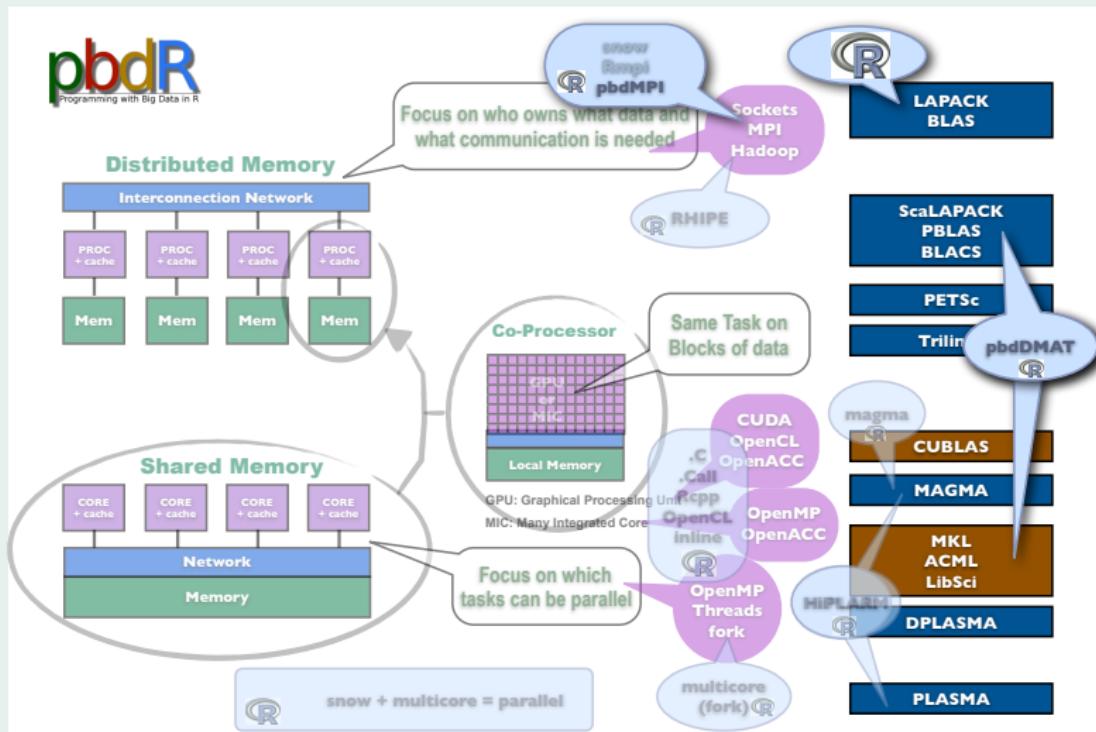


A Quick Overview of Parallel Software



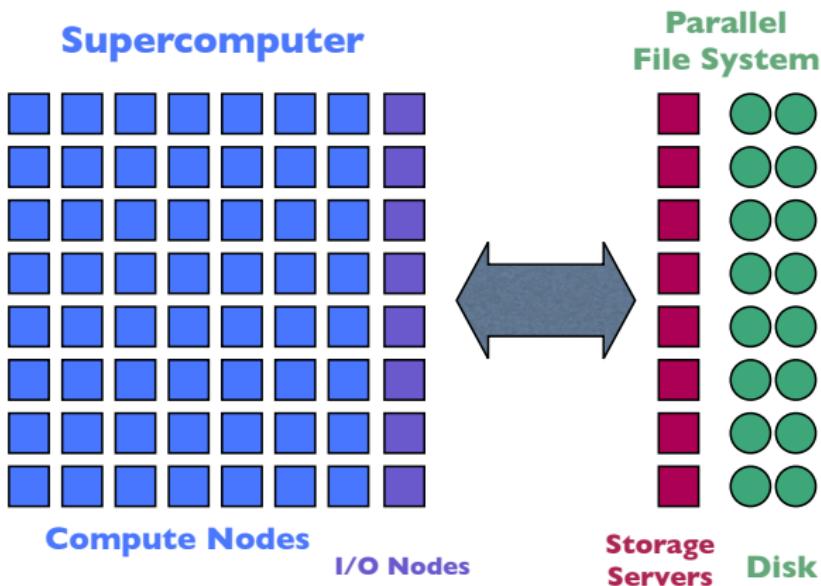
A Quick Overview of Parallel Software

pbdR Interfaces to Scalable HPC Libraries



A Quick Overview of Parallel Software

Building R Infrastructure for Supercomputers



Contents

2 pbdR

- The pbdR Project
 - Using pbdR

Programming with Big Data in R (pbdR)

Striving for *Productivity, Portability, Performance*

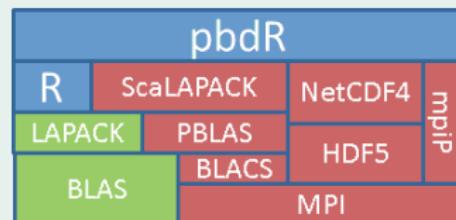
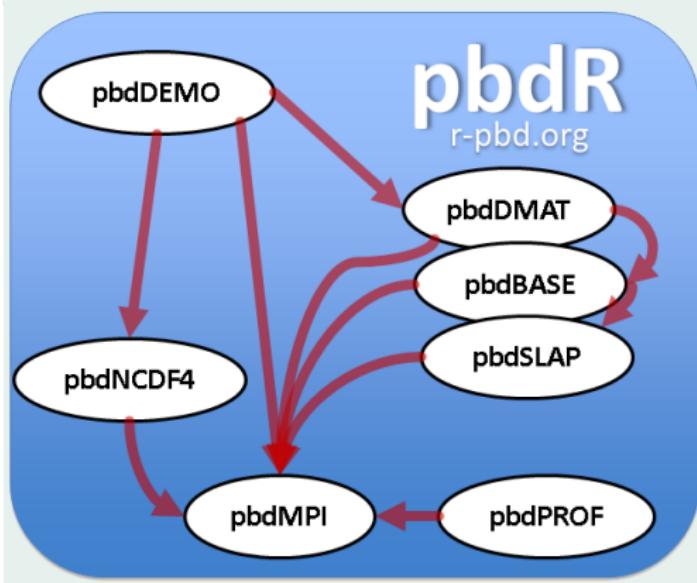


- Free^a R packages.
 - Bridging high-performance compiled code with high-productivity of R
 - Scalable, big data analytics.
 - Offers implicit and explicit parallelism.
 - Methods have syntax *identical* to R.

^aMPL, BSD, and GPL licensed

The pbdR Project

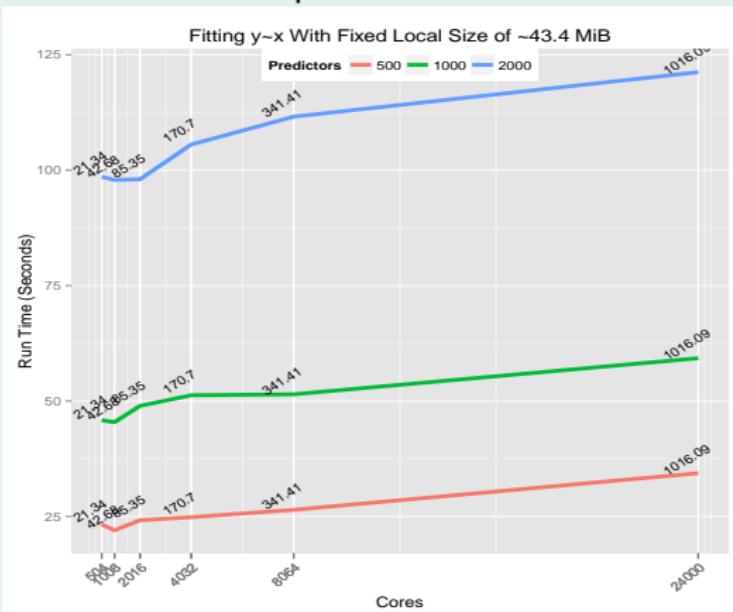
pbdR Packages



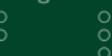
The pbdR Project

Distributed Matrices and Statistics with pbdDMAT

Least Squares Benchmark



```
x <- ddmatrix("rnorm", nrow=m, ncol=n)
y <- ddmatrix("rnorm", nrow=m, ncol=1)
mdl <- lm.fit(x=x, y=y)
```



The pbdR Project

Profiling with pbdPROF

```
R CMD INSTALL
  pbdMPI_0.2-1.tar.gz \
  --configure-args= \
  "--enable-pbdPROF"
```

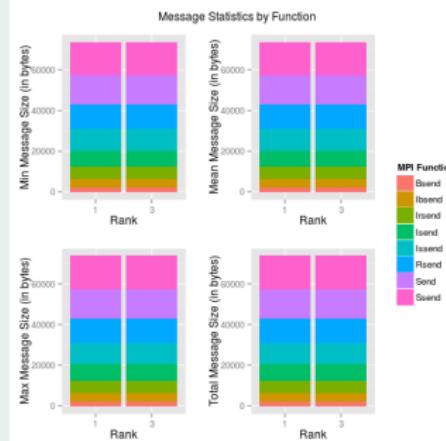
1. Rebuild pbdR packages

```
mpirun -np 64 Rscript
  my_script.R
```

2. Run code

```
library(pbdPROF)
prof <- read.prof(
  "profiler_output.mpiP")
plot(prof)
```

Publication-quality graphs



Using pbdR

pbdR Paradigms

pbdR programs are R programs!

Differences:

- Batch execution (non-interactive).
- Parallel code utilizes Single Program/Multiple Data (SPMD) style
- Emphasizes data parallelism.

Using pbdR

Batch Execution

- Running a serial R program in batch:

```
1 Rscript my_script.r
```

or

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

- Running a parallel (with MPI) R program in batch:

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



Using pbdR

Single Program/Multiple Data (SPMD)

- SPMD is a programming *paradigm*.
- Not to be confused with SIMD.

Paradigms

Programming models

e.g. Procedural, OOP,
Functional, SPMD, ...

SIMD

Hardware instructions

e.g. MMX, SSE, ...

SPMD is arguably the simplest extension of serial programming.

Using pbdR

Single Program/Multiple Data (SPMD)

- 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.

Contents

3 Introduction to pbdMPI

- Managing a Communicator
- Reduce, Gather, Broadcast, and Barrier
- Other pbdMPI Tools

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.

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

```
1 library(pbdMPI, quietly = 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
```

Managing a Communicator

Quick Example 2

Hello World: 2_hello.r

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

Execute this script via:

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

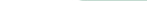
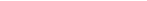
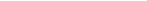
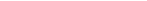
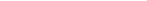
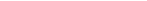
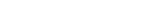
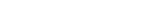
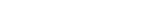
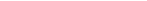
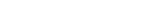
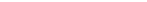
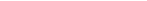
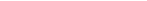
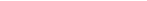
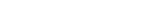
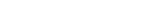
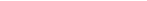
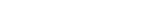
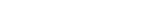
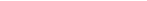
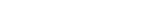
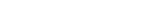
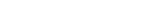
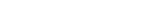
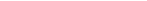
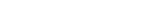
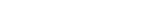
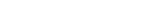
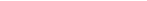
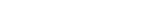
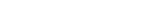
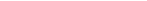
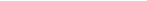
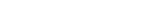
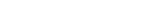
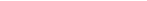
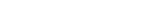
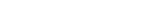
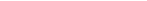
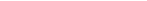
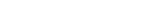
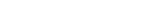
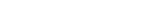
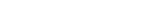
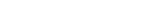
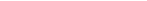
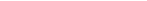
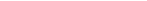
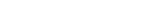
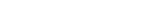
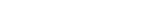
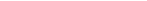
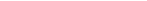
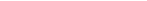
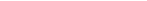
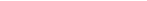
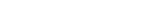
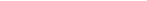
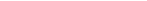
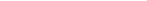
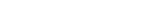
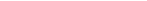
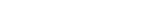
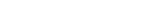
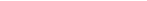
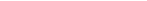
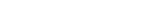
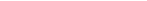
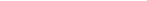
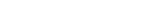
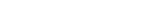
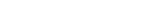
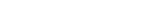
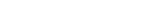
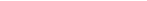
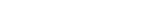
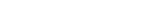
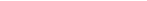
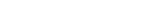
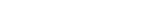
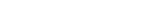
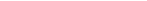
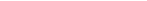
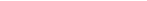
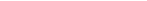
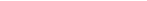
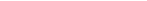
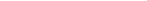
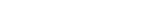
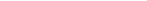
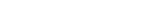
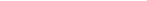
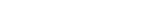
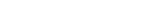
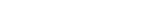
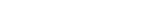
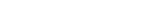
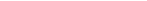
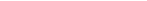
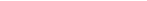
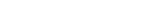
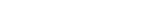
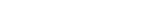
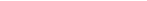
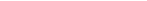
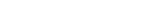
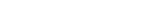
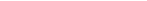
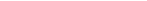
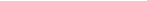
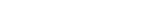
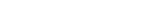
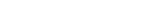
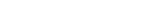
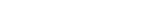
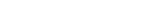
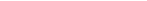
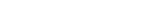
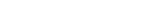
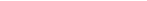
Sample Output:

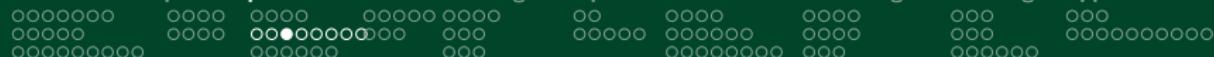
```
1 COMM.RANK = 0
2 [1] "Hello, world"
3 [1] "Hello again"
4 [1] "Hello again"
```

Reduce, Gather, Broadcast, and Barrier

MPI Operations

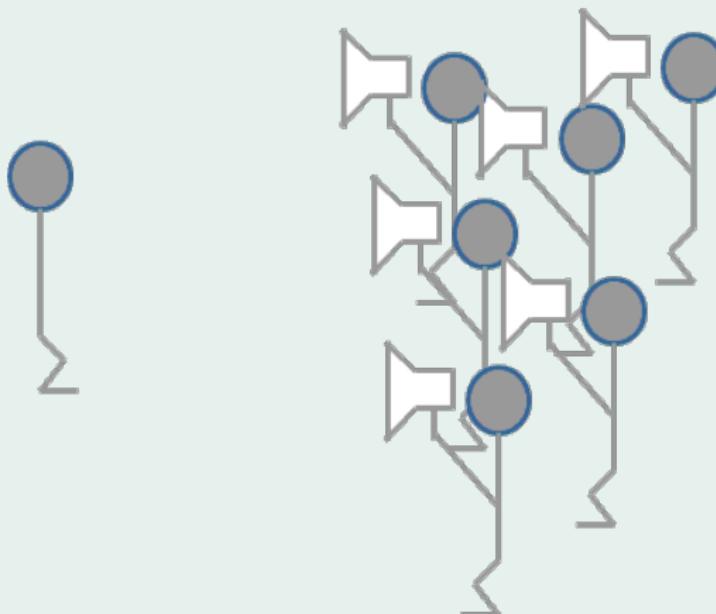
- ① Reduce
- ② Gather
- ③ Broadcast
- ④ Barrier



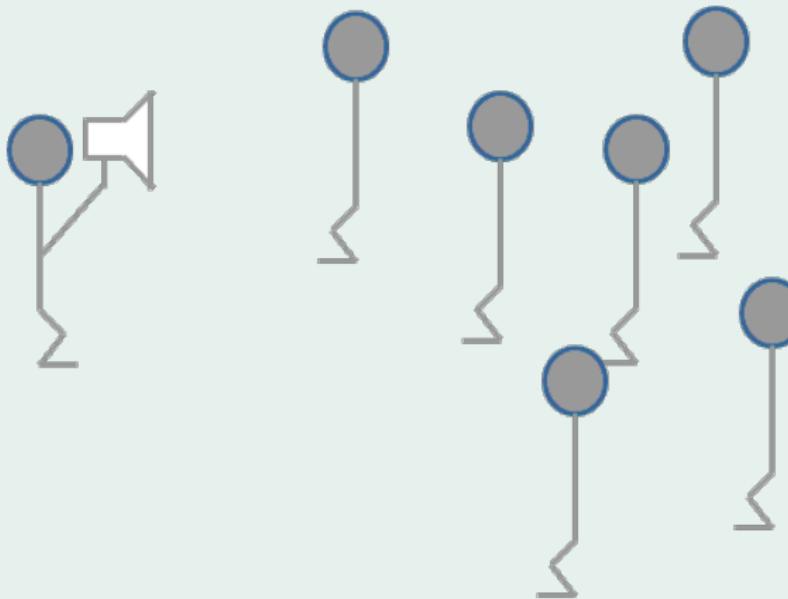


Reduce, Gather, Broadcast, and Barrier

Gather — Many-to-one

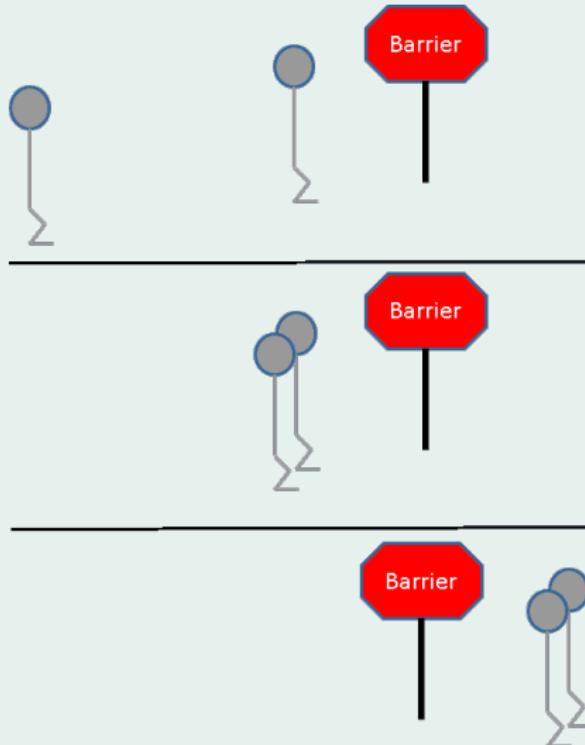


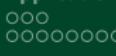
Broadcast — One-to-many



Reduce, Gather, Broadcast, and Barrier

Barrier — Synchronization





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()`

Reduce, Gather, Broadcast, and Barrier

Quick Example 3

Reduce and Gather: 3_gt.r

```
1 library(pbdMPI, quietly=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
```

Reduce, Gather, Broadcast, and Barrier

Quick Example 4

Broadcast: 4_bcast.r

```
1 library(pbdMPI, quietly=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
```

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



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).

Other pbdMPI Tools

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

```

Execute this script via:

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

Sample Output:

	min	mean	max
1	0.17	0.173	0.176

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()

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

4 The Generalized Block Distribution

- The GBD Data Structure
- Example GBD Distributions

The GBD Data Structure

Distributing Data

Problem: How to distribute the data

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

The GBD Data Structure

Distributing a Matrix Across 4 Processors: Block Distribution

Data	Processors
$x_{1,1}$	0
$x_{2,1}$	1
$x_{3,1}$	2
$x_{4,1}$	3
$x_{5,1}$	
$x_{6,1}$	
$x_{7,1}$	
$x_{8,1}$	
$x_{9,1}$	
$x_{10,1}$	
$x = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \\ \hline x_{4,1} & x_{4,2} & x_{4,3} \\ x_{5,1} & x_{5,2} & x_{5,3} \\ x_{6,1} & x_{6,2} & x_{6,3} \\ \hline x_{7,1} & x_{7,2} & x_{7,3} \\ x_{8,1} & x_{8,2} & x_{8,3} \\ x_{9,1} & x_{9,2} & x_{9,3} \\ \hline x_{10,1} & x_{10,2} & x_{10,3} \end{bmatrix}_{10 \times 3}$	

The GBD Data Structure

Distributing a Matrix Across 4 Processors: Local Load Balance

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

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



The GBD Data Structure

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

- ➊ GBD is *distributed*. No processor owns all the data.
- ➋ GBD is *non-overlapping*. Rows uniquely assigned to processors.
- ➌ GBD is *row-contiguous*. If a processor owns one element of a row, it owns the entire row.
- ➍ GBD is globally *row-major*, locally *column-major*.
- ➎ GBD is often *locally balanced*, where each processor owns (almost) the same amount of data. But this is not required.
- ➏ 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.
- ➐ GBD is (relatively) easy to understand, but can lead to bottlenecks if you have many more columns than rows.

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$
$x_{6,1}$	$x_{6,2}$	$x_{6,3}$
$x_{7,1}$	$x_{7,2}$	$x_{7,3}$
$x_{8,1}$	$x_{8,2}$	$x_{8,3}$
$x_{9,1}$	$x_{9,2}$	$x_{9,3}$
$x_{10,1}$	$x_{10,2}$	$x_{10,3}$

The GBD Data Structure

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.

Example GBD Distributions

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

Example GBD Distributions

Understanding GBD: Load Balanced GBD

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

Processors = 0 1 2 3 4 5

Example GBD Distributions

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

Contents

5 Basic Statistics Examples

- pbdMPI Example: Monte Carlo Simulation
- pbdMPI Example: Sample Covariance
- pbdMPI Example: Linear Regression

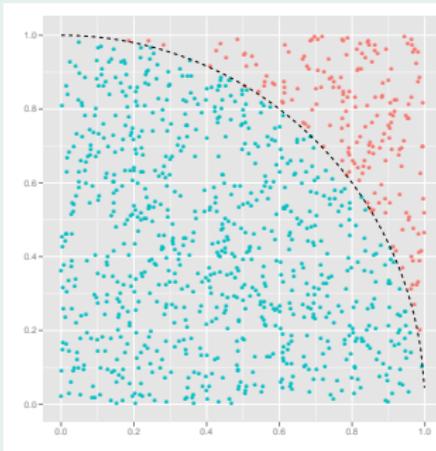


pbdMPI Example: Monte Carlo Simulation

Example 1: Monte Carlo Simulation

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

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





pbdMPI Example: Monte Carlo Simulation

Example 1: Monte Carlo Simulation GBD Algorithm

- ① Let n be big-ish; we'll take $n = 50,000$.
- ② Generate an $n \times 2$ matrix x of standard uniform observations.
- ③ Count the number of rows satisfying $x^2 + y^2 \leq 1$
- ④ Ask everyone else what their answer is; sum it all up.
- ⑤ Take this new answer, multiply by 4 and divide by n
- ⑥ If my rank is 0, print the result.



pbdMPI Example: Monte Carlo Simulation

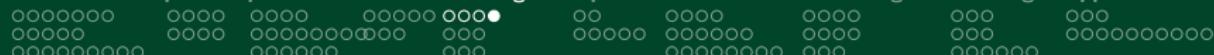
Example 1: Monte Carlo Simulation Code

Serial Code

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

Parallel Code

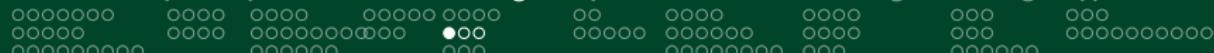
```
1 library(pbdMPI, quiet = TRUE)
2 init()
3 comm.set.seed(diff=TRUE)
4
5 N.gbd <- 50000 / comm.size()
6 X.gbd <- matrix(runif(N.gbd * 2), ncol = 2)
7 r.gbd <- sum(rowSums(X.gbd^2) <= 1)
8 r <- allreduce(r.gbd)
9 PI <- 4*r/(N.gbd * comm.size())
10 comm.print(PI)
11
12 finalize()
```



pbdMPI Example: Monte Carlo Simulation

Note

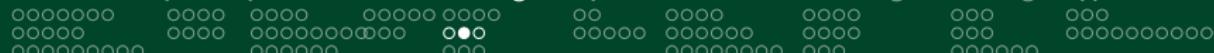
For the remainder, we will exclude loading, init, and finalize calls.



pbdMPI Example: Sample Covariance

Example 2: Sample Covariance

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



pbdMPI Example: Sample Covariance

Example 2: Sample Covariance GBD Algorithm

- ① Determine the total number of rows N .
- ② Compute the vector of column means of the full matrix.
- ③ Subtract each column's mean from that column's entries in each local matrix.
- ④ Compute the crossproduct locally and reduce.
- ⑤ Divide by $N - 1$.



pbdMPI Example: Sample Covariance

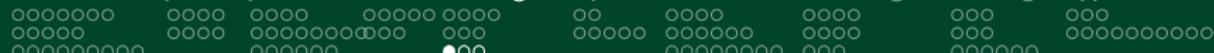
Example 2: Sample Covariance Code

Serial Code

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

Parallel Code

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



pbdMPI Example: Linear Regression

Example 3: Linear Regression

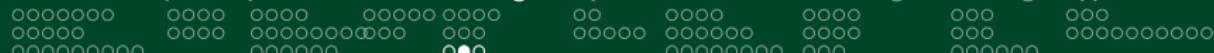
Find β such that

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

When X is full rank,

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





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 $A^{-1} * B$



pbpdMPI 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

6 Data Input

- Serial Data Input
 - Parallel Data Input

Serial Data Input

Separate manual: <http://r-project.org/>

- `scan()`
- `read.table()`
- `read.csv()`
- `socket`

Serial Data Input

CSV Data: Read Serial then Distribute

Listing:

```
1 library(pbdDMAT)
2 if(comm.rank() == 0) { # only read on process 0
3   x <- read.csv("myfile.csv")
4 } else {
5   x <- NULL
6 }
7
8 dx <- as.ddmatrix(x)
```



Parallel Data Input

New Issues

- How to read in parallel?
- CSV, SQL, NetCDF4, HDF, ADIOS, custom binary
- How to partition data across nodes?
- How to structure for scalable libraries?
- Read directly into form needed or restructure?
- ...
- A lot of work needed here!



CSV Data

Serial Code

```
1 d <- read.csv('x.csv')
```

Parallel Code 0_readcsv.r

```
1 library(pbdDEMO, quiet = TRUE)
2 init.grid()
3 dx <- read.csv.ddmatrix("x.csv", header=TRUE,
4                         sep=',', nrows=10, ncols=10,
5                         num.rdrs=2, ICTXT=0)
6 comm.print(dx)
7 finalize()
```

Parallel Data Input

CSV Data

NetCDF4 Files

```
1 ##### Must determine who will read what portion(s) and how
2 # to assemble
3 #
4 ##### parallel read after determining st and co
5 nc <- nc_open_par(file.name)
6 #
7 nc_var_par_access(nc, "TREFHT")
8 new.X.gbdc <- ncvar_get(nc, "TREFHT", start = st, count
9 # = co)
10 nc_close(nc)
11 finalize()
```





Parallel Data Input

CSV Data

3d Block Binary Reader

```

1 library(pbdDMAT, quiet = TRUE)
2 init.grid()
3
4 ## Raw data dimensions in file
5 data.dim <- c(2048, 2048, 2048)
6
7 ## global subcube definition
8 g.start <- c(1, 1, 513)
9 g.dim <- c(64, 64, 1024)
10
11 ## local dimension and start
12 my.dim <- g.dim / c(1, 1, comm.size())
13 my.start <- g.start + c(0, 0, comm.rank()*my.dim[3])
14
15 size <- 4 # file is single precision floats
16 project <-
17   "/lustre/atlas/proj-shared/stf006/d7r/inv_cascade"
18 filer <- paste(project,
19   "rot_abc_2048_kf50/outs/vx.061.out", sep="/")
20
21 vx <- block3d.read(filer, data.dim, my.start, my.dim,
22   size)
23
24 save.file <- paste("xyz.RData", comm.rank(), sep="")
25 save(vx, file=save.file)
26
27 ## make a ddmatrix from local data
28 ## reshape 3d array into a matrix for PCA (EOF)
29 ## computation
30 ## first two dimensions become rows and third becomes
31 ## columns
32
33 ## local reshape dimensions
34 my.nrow <- prod(my.dim[1:2])
35 my.ncol <- my.dim[3]
36 ldim <- c(my.nrow, my.ncol)
37
38 ## global reshape dimensions
39 g.nrow <- prod(g.dim[1:2])
40 g.ncol <- g.dim[3]
41 gdim <- c(g.nrow, g.ncol)
42
43 ## now reshape local
44 X <- matrix(vx, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)
45
46 ## glue local pieces into a ddmatrix
47 X <- new("ddmatrix", Data=X, dim=gdim, ldim=ldim,
48   bldim=ldim, ICTXT=1)
49
50 ## transform to 2d block cyclic
51 X <- redistribute(X, bldim=c(8,8), ICTXT=0)

```



Parallel Data Input

CSV Data

3d Block Binary Reader

```

1 block3d.read <- function(file, data.dim, my.start,
2   my.dim, size=4) {
3   con.x <- file(file, "rb", blocking=TRUE)
4   if(isSeekable(con.x)) {
5
6     start <- sum((my.start - 1) * c(1,
7       cumprod(data.dim)[-length(data.dim)]))
8
9     x <- rep(NA, prod(my.dim))
10
11    block <- 1:my.dim[1]
12
13    for(j in 1:my.dim[3]) {
14     sofar <- 0
15      for(i in 1:my.dim[2]) {
16        seek(con.x, where=start, rw="read",
17          origin="start")
18        x[block] <- readBin(con=con.x,
19          what="numeric",
20          n=my.dim[1],
21          size=size)
22        block <- block + my.dim[1]
23
24        start <- start + data.dim[1]*size
25        sofar <- sofar + data.dim[1]*size
26      }
27      start <- start - sofar +
28        data.dim[1]*data.dim[2]*size
29    }
30  }
31 else {
32   x <- NULL
33   comm.print(paste("Sorry ...", file, "not
34   seekable!"))
35 }
36 close(con.x)
37 x
38 }
```

Contents

7 Introduction to pbdDMAT and the DMAT Structure

- Introduction to Distributed Matrices
- DMAT Distributions
- pbdDMAT

Introduction to Distributed Matrices

Distributed Matrices

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

Distributed matrices \implies Handle Bigger data

Introduction to Distributed Matrices

Distributed Matrices



(a) Block



(b) Cyclic



(c) Block-Cyclic

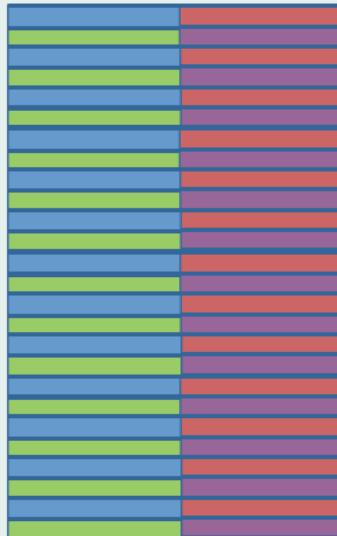
Figure: Matrix Distribution Schemes

Introduction to Distributed Matrices

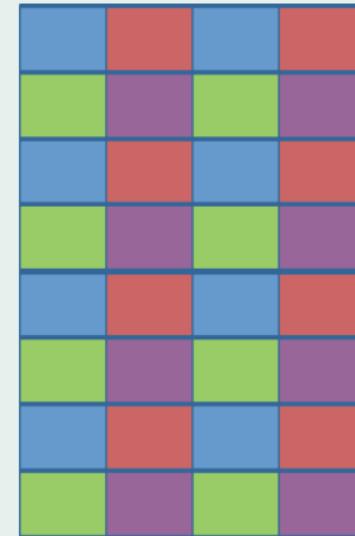
Distributed Matrices



(a) 2d Block

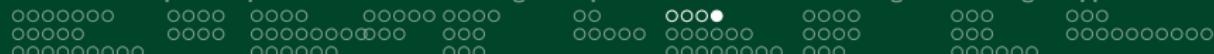


(b) 2d Cyclic



(c) 2d Block-Cyclic

Figure: Matrix Distribution Schemes Onto a 2-Dimensional Grid



Introduction to Distributed Matrices

Processor Grid Shapes

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

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

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

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

(a) 1×6 (b) 2×3 (c) 3×2 (d) 6×1

Table: Processor Grid Shapes with 6 Processors

DMAT Distributions

Understanding Dmat: Global Matrix

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

DMAT Distributions

DMAT: 1-dimensional Row Block

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

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

DMAT Distributions

DMAT: 2-dimensional Row Block

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

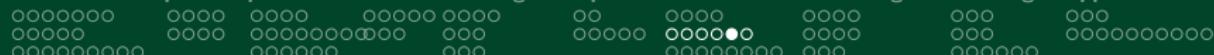
$$\text{Processor grid} = \begin{vmatrix} 0 & 1 \\ 2 & 3 \end{vmatrix} = \begin{vmatrix} (0,0) & (0,1) \\ (1,0) & (1,1) \end{vmatrix}$$

DMAT Distributions

DMAT: 1-dimensional Row Cyclic

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

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



DMAT Distributions

DMAT: 2-dimensional Row Cyclic

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

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

DMAT Distributions

DMAT: 2-dimensional Block-Cyclic

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

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



pbdDMAT

The DMAT Data Structure

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



pbdDMAT

DMAT: 2-dimensional Block-Cyclic with 6 Processors

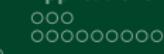
$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\ x_{41} & x_{42} & x_{43} & x_{44}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

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

Understanding DMAT: Local View

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

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



pbdDMAT

The DMAT Data Structure

- ➊ DMAT is *distributed*. No one processor owns all of the matrix.
- ➋ DMAT is *non-overlapping*. Any piece owned by one processor is owned by no other processors.

- ➌ DMAT can be row-contiguous or not, depending on the processor grid and blocking factor used.
- ➍ DMAT is locally column-major and globally, it depends...
- ➎ GBD is a generalization of the one-dimensional block DMAT distribution. Otherwise there is no relation.
- ➏ DMAT is confusing, but very robust.

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}



pbdDMAT

Pros and Cons of This Data Structure

Pros

- Robust for matrix computations.

Cons

- Confusing layout.

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)
```

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.

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

8 Examples Using pbdDMAT

- Manipulating DMAT Objects
 - Statistics Examples with pbdDMAT
 - RandSVD





Manipulating DMAT Objects

Example 1: DMAT Construction

Generate a global matrix and distribute it

```
1 library(pbdDMAT, quiet=TRUE)
2 init.grid()
3
4 # Common global on all processors --> distributed
5 x <- matrix(1:100, nrow=10, ncol=10)
6 x.dmat <- as.ddmatrix(x)
7
8 x.dmat
9
10 # Global on processor 0 --> distributed
11 if (comm.rank()==0){
12   y <- matrix(1:100, nrow=10, ncol=10)
13 } else {
14   y <- NULL
15 }
16 y.dmat <- as.ddmatrix(y)
17
18 y.dmat
19
20 finalize()
```

Execute this script via:

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

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,
12   mean=10, sd=100)
13 rand.dmat
14 finalize()
```

Execute this script via:

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



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
```

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:

Statistics Examples with pbdDMAT

Execute this script via:

```
1 % mpirun -np 2 Rscript 4_convert.r  
2 %
```

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)
```



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)
```



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
```

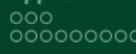
Sample Output:

1 Cols: 221
2 %Cols: 0.884

Statistics Examples with pbdDMAT

Distributed Matrices

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



RandSVD

Randomized SVD¹

PROTOTYPE FOR RANDOMIZED SVD

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

Stage A:

- 1 Generate an $n \times 2k$ Gaussian test matrix Ω .
- 2 Form $Y = (AA^*)^q \Omega$ 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 $m \times l$ orthonormal matrix Q whose range approximates the range of A .

- 1 Draw an $n \times l$ standard Gaussian matrix Ω .
- 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 }
```

¹Halko 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

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 }
```

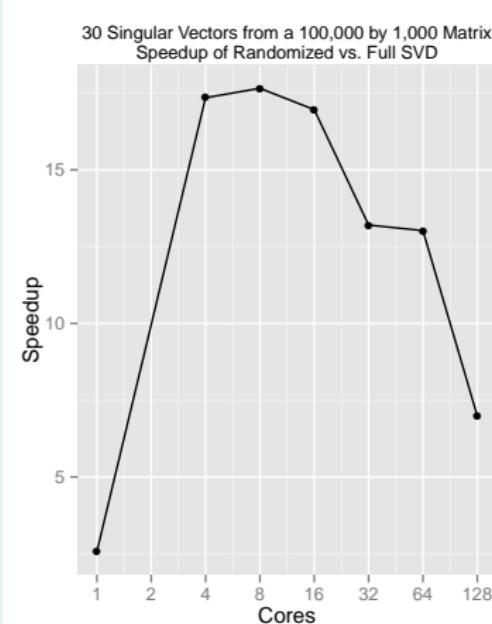
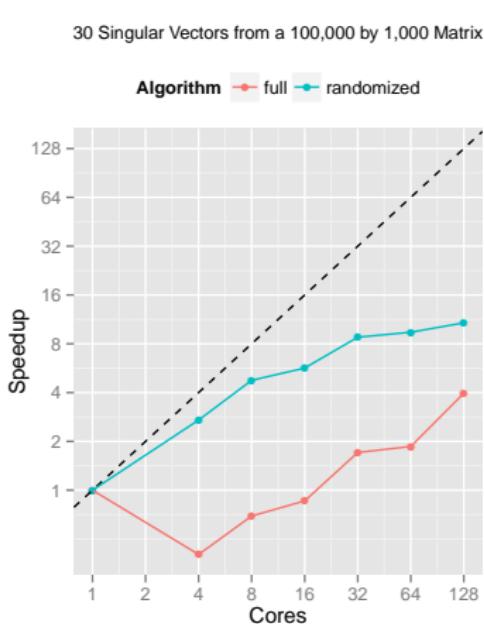
Parallel pbdR

```

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



Contents

9 Profiling

- Profiling
 - Installing pbdPROF
 - Example



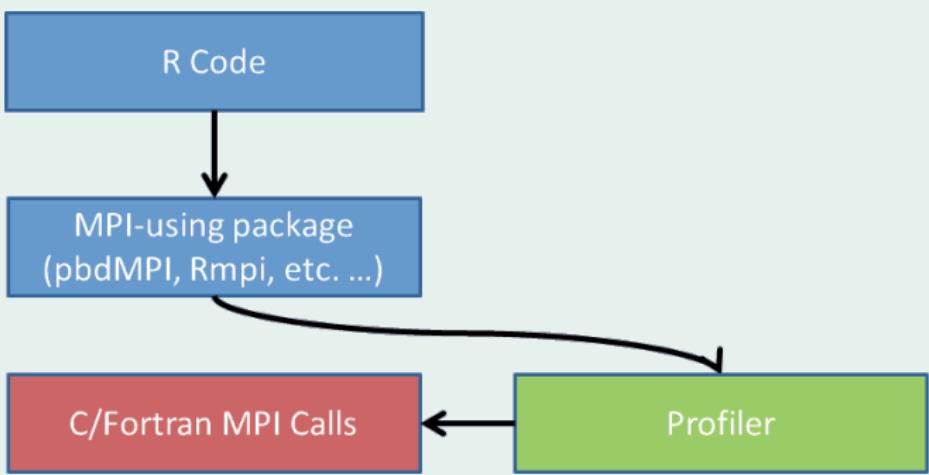
Introduction to pbdPROF

- Successful Google Summer of Code 2013 project.
 - Available on the CRAN.
 - Enables profiling of MPI-using R scripts.
 - **pbdR** packages officially supported; can work with others. . .
 - Also reads, parses, and plots profiler outputs.

Profiling

How it works

MPI calls get hijacked by profiler and logged:



Profiling

Introduction to **pbdPROF**

- Currently supports the profilers **fpm MPI** and **mpiP**.
- **fpm MPI** is distributed with **pbdPROF** and installs easily, but offers minimal profiling capabilities.
- **mpiP** is fully supported also, but harder to install.

Installing pbdPROF

- ➊ Build **pbdPROF**.
- ➋ Rebuild **pbdMPI** (linking with **pbdPROF**).
- ➌ Run your analysis as usual.
- ➍ Interactively analyze profiler outputs with **pbdPROF**.



Installing pbdPROF

Build pbdPROF

```
R CMD INSTALL pbdPROF_0.2-1.tar.gz
```

- The above installs **fpm MPI**.
- **mpiP** can be used if you have a system installation available.
- See package vignette for more details and troubleshooting.

Installing pbdPROF

Rebuild pbdMPI

```
R CMD INSTALL pbdMPI_0.2-2.tar.gz  
--configure-args="--enable-pbdPROF"
```

- Any package which explicitly links with an MPI library must be rebuilt in this way (**pbdMPI**, **Rmpi**, ...).
- Other **pbdR** packages link with **pbdMPI**, and so do not need to be rebuilt.
- See **pbdPROF** vignette if something goes wrong.

Example

Example Script

my_svd.r

```
1 library(pbdMPI, lib.loc = "~/R/prof", quietly=TRUE)
2 library(pbdDMAT, quietly=T)
3 init.grid()
4
5 n <- 1000
6 x <- ddmatrix("rnorm", n, n)
7
8 asdf <- La.svd(x)
9
10
11 finalize()
```



Example

Example Script

Run example with 4 ranks:

```
$ mpirun -np 4 Rscript my_svd.r
mpiP:
mpiP: mpiP V3.4.0 (Build Feb 14 2014/13:55:39)
mpiP: Direct questions and errors to
      mpip-help@lists.sourceforge.net
mpiP:
Using 2x2 for the default grid size

mpiP:
mpiP: Storing mpiP output in [./R.4.28812.1.mpiP].
mpiP:
```



Example

Read Profiler Data into R

Interactively (or in batch) Read in Profiler Data

```
1 library(pbdPROF)
2 prof.data <- read.prof("R.4.28812.1.mpiP")
```

Partial Output of Example Data

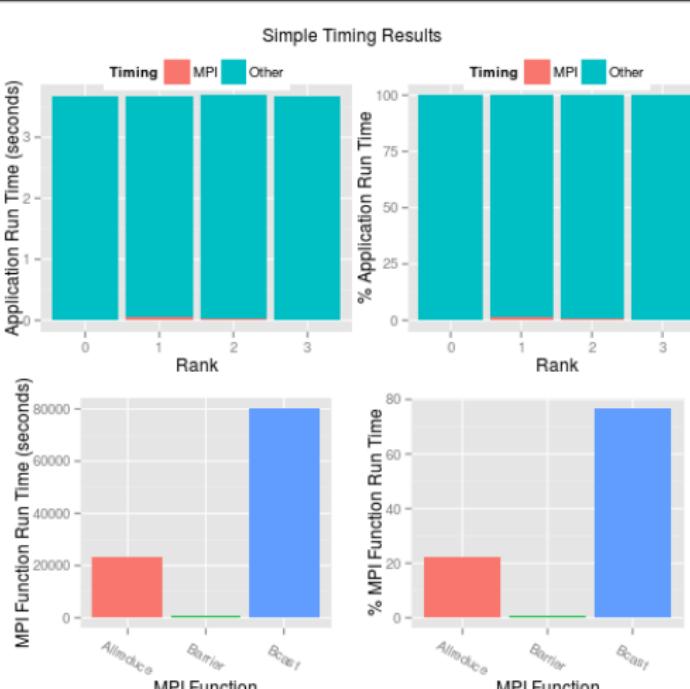
```
> prof.data
An mpip profiler object:
[[1]]
  Task AppTime MPITime MPI.
1     0    3.68  0.00816  0.22
2     1    3.68  0.04890  1.33
3     2    3.69  0.03850  1.04
4     3    3.68  0.00904  0.25
5     *   14.70  0.10500  0.71

[[2]]
  ID Lev File.Address Line_Parent_Funct MPI_Call
1   1   0 1.400699e+14           [unknown] Allreduce
2   2   0 1.400699e+14           [unknown] Allreduce
```

Example

Generate plots

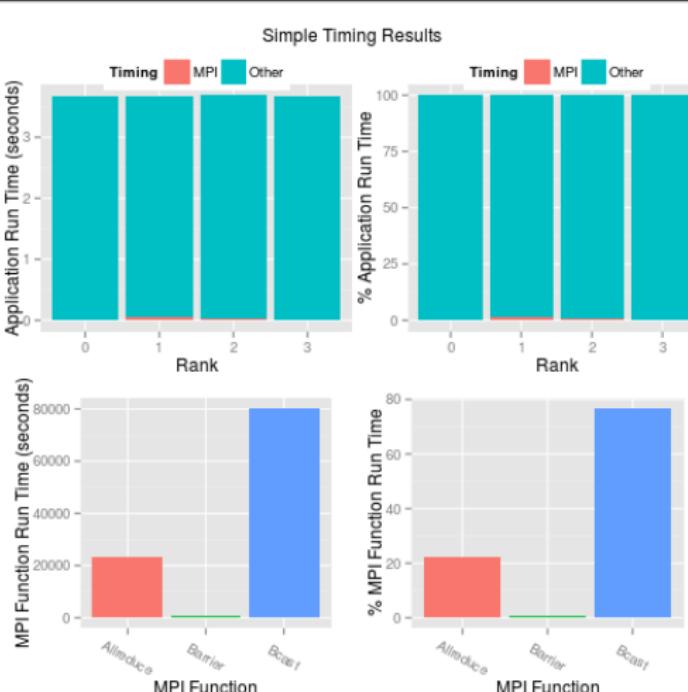
```
1 | plot(prof.data)
```



Example

Generate plots

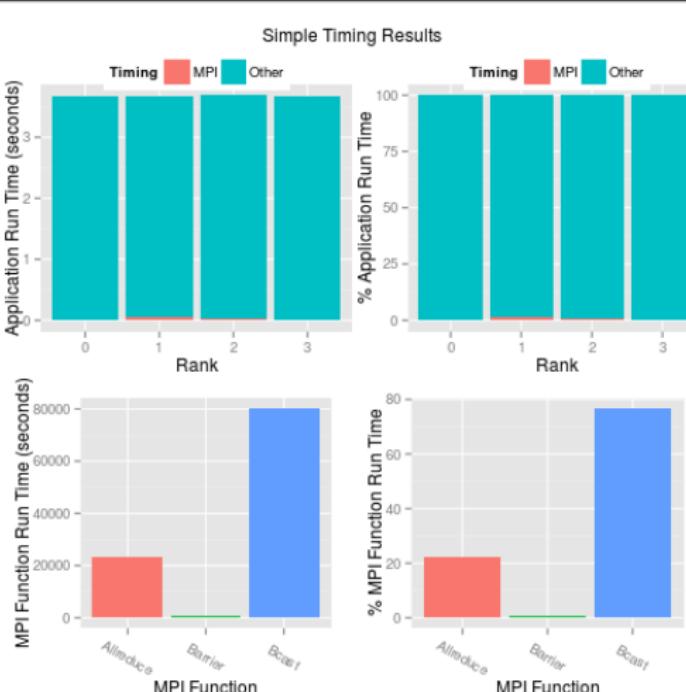
```
1 | plot(prof.data, plot.type="stats1")
```



Example

Generate plots

```
1 plot(prof.data, plot.type="messages1")
```



Contents

10 Example Applications

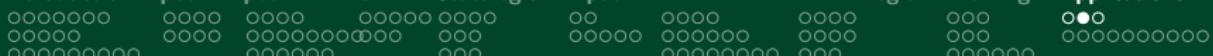
- Principal Components Analysis
- Parallel Plot Ensembles

Principal Components Analysis

Empirical Orthogonal Functions in Climate Analysis

- Computation and volume rendering of large-scale EOF coherent modes in rotating turbulent flow data, AGU Fall Meeting, December 2013





Principal Components Analysis

Coherent Modes in Turbulent Flow

Get and Redistribute the Data

```

1 library(pbdDMAT, quiet = TRUE)
2 init.grid()
3
4 ## load local data (file assumes 4 processors!)
5 g.dim <- c(64, 64, 1024)
6 my.dim <- g.dim / c(1, 1, comm.size())
7 save.file <- paste("xyz.RData", comm.rank(), sep=".") #
   assumes 4 processors!
8 load(save.file)
9
10 ## reshape 3d array into a matrix for PCA (EOF)
    computation
11 ## first two dimensions become rows and third becomes
    columns
12 ## local reshape dimensions
13 my.nrow <- prod(my.dim[1:2])
14 my.ncol <- my.dim[3]
15 ldim <- c(my.nrow, my.ncol)
16
17 ## global reshape dimensions
18 g.nrow <- prod(g.dim[1:2])
19 g.ncol <- g.dim[3]
20 gdim <- c(g.nrow, g.ncol)
21
22 ## now reshape local
23 X <- matrix(vx, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)
24 Y <- matrix(vy, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)
25 Z <- matrix(vz, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)
26
27 ## glue local pieces into a ddmatrix
28 X <- new("ddmatrix", Data=X, dim=gdim, ldim=ldim,
   bldim=ldim, ICTXT=1)
29 Y <- new("ddmatrix", Data=Y, dim=gdim, ldim=ldim,
   bldim=ldim, ICTXT=1)
30 Z <- new("ddmatrix", Data=Z, dim=gdim, ldim=ldim,
   bldim=ldim, ICTXT=1)
31
32 ## transform to 2d block cyclic
33 X <- redistribute(X, bldim=c(8,8), ICTXT=0)
34 Y <- redistribute(Y, bldim=c(8,8), ICTXT=0)
35 Z <- redistribute(Z, bldim=c(8,8), ICTXT=0)

```

Principal Components Analysis

Coherent Modes in Turbulent Flow

Compute PCA and do Scree Plot (0_pca.r)

```

1 E <- sqrt(X^2 + Y^2 + Z^2) # energy from velocity
2 E.pca <- prcomp(x=E, retx=TRUE, scale=FALSE)
3
4 # plot using one process
5 if(comm.rank() == 0)
6 {
7   ## scree plot for first 50 components
8   variance <- E.pca$sdev^2 # note: all own sdev
9   proportion <- variance[1:50]/sum(variance)
10  cumulative <- cumsum(proportion)
11  component <- 1:length(proportion)
12  png("scree.png")
13  plot(component, cumulative, ylim=c(0,1))
14  points(component, proportion, type="h", col="blue")
15  dev.off()
16 }
17
18 finalize()

```

Parallel Plot Ensembles

How can we plot in parallel?

- Several plots, one on each processor
- One plot by several processors



Parallel Plot Ensembles

Plots in parallel

```
png.slice  
1 png.slice <- function(x, g.dim, lab="slice", title=lab,  
2   work.dir="", zero.center=TRUE, most.positive=TRUE)  
3 {  
4   X <- array(as.vector(x), dim=g.dim)  
5  
6   ## prepare zero centered topo.colors  
7   if(zero.center)  
8   {  
9     . . .  
10    }  
11  else  
12    zlim <- range(X)  
13  
14  ## set most positive (for unique up to sign)  
15  if(most.positive)  
16  {  
17    . . .  
18  
19  ## make png file  
20  file <- paste(work.dir, lab, "-r", comm.rank(),  
21    ".png", sep="")  
22  png(file)  
23  image(x=1:g.dim[1], y=1:g.dim[2], z=X,  
24    col=topo.colors(40), useRaster=TRUE, asp=1,  
25    xlim=c(1, g.dim[1] + 1), ylim=c(1, g.dim[2] + 1),  
26    zlim=zlim)  
27  title(title)  
28  ret <- dev.off()  
29  invisible(ret)  
30 }
```

Parallel Plot Ensembles

Plots in parallel

Get and Redistribute the Data

```

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

## set global and local dimensions
g.dim <- c(64, 64, 1024)
my.dim <- g.dim / c(1, 1, comm.size())
load(save.file)      # gets vx vector

## reshape 3d array into a matrix
## first two dimensions become rows and third becomes
## columns

## local reshape dimensions
my.nrow <- prod(my.dim[1:2])
my.ncol <- my.dim[3]
ldim <- c(my.nrow, my.ncol)

## global reshape dimensions
g.nrow <- prod(g.dim[1:2])
g.ncol <- g.dim[3]
gdim <- c(g.nrow, g.ncol)

## now reshape local
X <- matrix(vx, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)

## glue local pieces into a ddmatrix
X <- new("ddmatrix", Data=X, dim=gdim, ldim=ldim,
        blim=ldim, ICTXT=1)

## transform to 2d block cyclic
X <- redistribute(X, bldim=c(8,8), ICTXT=0)

## Plot few columns in parallel
. .
finalize()

```



Parallel Plot Ensembles

Plots in parallel

Make comm.size() plots in parallel

```
1 step <- 5
2 max.plots <- min(20, ncol(X) %/% step)
3 last.plot <- 1 - step
4 time <- comm.timer(
5 for(i in 1:max.plots)
6 {
7     now.plots <- last.plot + step*(1:comm.size())
8     my.col <- gather.col(X[, now.plots])
9     lab <- paste("col", lead0(now.plots[comm.rank()
10        + 1]), sep="")
11     png.slice(my.col, g.dim[1:2], lab)
12     last.plot <- now.plots[length(now.plots)]
13 }
```

Parallel Plot Ensembles

Plots in parallel

gather.col First Attempt (1_plot.r)

```
1 gather.col <- function(x, num=min(ncol(x), comm.size()))
2 {
3     ## gather complete columns of a global array to
4     ## different ranks
5     my.local <- as.vector(x[, comm.rank() + 1],
6                           proc.dest=comm.rank())
7     my.local
8 }
```

Parallel Plot Ensembles

Plots in parallel

gather.col Second Attempt (2_plot.r)

```
1 gather.col <- function(x, num=min(ncol(x), comm.size()))
2 {
3     ## gather complete columns of a global array to
4     ## different ranks
5     my.local <- NULL
6     for(i in 1:num)
7     {
8         ## serial collection of unique data to each rank
9         local <- as.vector(x[, i])
10        if(comm.rank() + 1 == i) my.local <- local
11    }
12    my.local
13 }
```



Parallel Plot Ensembles

Plots in parallel

gather.col The Right Way (3_plot.r)

```

1 gather.col <- function(x, num=min(ncol(x), comm.size()))
2 {
3   ## gather complete columns of a global array to
4   ## different ranks
5   x.num <- x[, 1:num]
6   x.num <- as.colblock(x.num)
7
8   ## ScaLAPACK fix (a future release will automate)
9   if(ownany(x.num))
10     ret <- as.vector(submatrix(x.num))
11   else
12     ret <- NULL
13   ret
}

```



Parallel Plot Ensembles

Plots in parallel

Now Plot the PCA Components (4_plot.r)

```

1 E <- sqrt(X^2 + Y^2 + Z^2)
2
3 E.pca <- prcomp(x=E, retx=TRUE, scale=FALSE)
4
5 ## Use ranks 1 to n.pca to plot individual components in
   parallel
6 n.pca <- min(comm.size(), g.nrow)
7 my.col <- gather.col(E.pca$x, num=n.pca)
8
9 if(!is.null(my.col))
10 {
11     ## component plots on rank 1 to n.pca
12     lab <- paste("pc", comm.rank(), sep="")
13     title <- paste(lab, "sigma^2 =",
14                     variance[comm.rank() + 1])
15     png.slice(my.col, g.dim[1:2], lab, title=title,
                work.dir=work.dir)
}

```

Parallel Plot Ensembles

Simple Redistributions

- `as.block(dx, square.bldim = TRUE)`
- `as.rowblock(dx)`
- `as.colblock(dx)`
- `as.rowcyclic(dx, bldim = .BLDIM)`
- `as.colcyclic(dx, bldim = .BLDIM)`
- `as.blockcyclic(dx, bldim = .BLDIM)`

BLACS context (Processor Grid)

- `init.grid(P,Q)`
- `.ICTXT = 0` gives $P \times Q$
- `.ICTXT = 1` gives $PQ \times 1$
- `.ICTXT = 2` gives $1 \times PQ$

Parallel Plot Ensembles

Exercise: scripts/pbdDMAT/dmat_app

- Experiment with scripts 0_pca.r, 1_plot.r, 2_plot.r, 3_plot.r, 4_plot.r, 5ictxt.r, 6_ictxt.r, and 7_ictxt.r
- Experiment with other redistributions



Contents

11 Wrapup



The pbdR Project

- Our website: <http://r-pbd.org/>
- Email us at: RBigData@gmail.com
- Our google group: <http://group.r-pbd.org/>

Where to begin?

- The **pbdDEMO** package
<http://cran.r-project.org/web/packages/pbdDEMO/>
- The **pbdDEMO** Vignette: <http://goo.gl/HZkRt>



Thanks for coming!

Questions?



pbdR
Programming with Big Data in R

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