A Hasty Overview of pbdR, with an Application to Matrix Exponentiation

Drew Schmidt

April 7, 2014

http://r-pbd.org/NIMBioS

NIMBioS

Contents

1. The pbdR Project

Recall: Parallel R Packages

Shared Memory

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

Distributed

1. **Rmpi**
2. **RHIPE, RHadoop**
3. **pbdR**

(and others. . .)

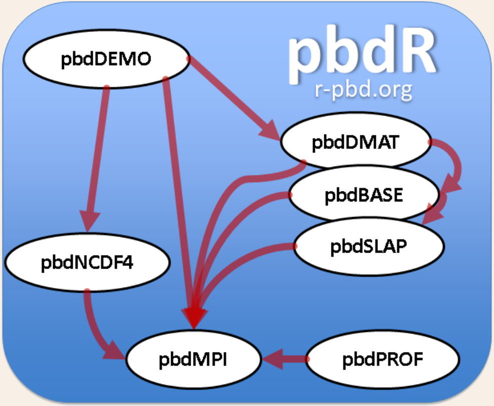
Programming with Big Data in R (pbdR)

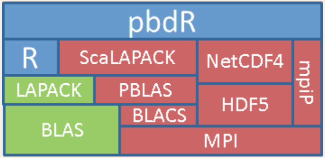
Striving for *Productivity, Portability, Performance*

- *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.

[a] MPL, BSD, and GPL licensed

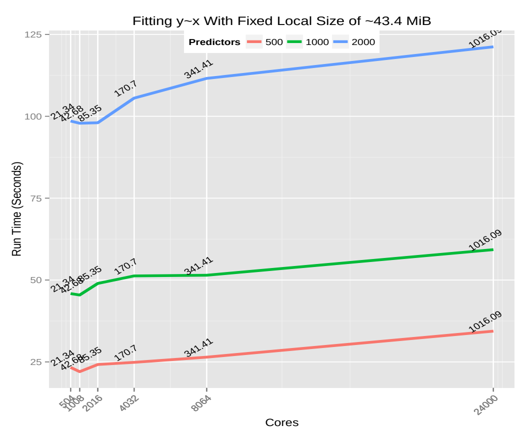
pbdR Packages





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)
```

Profiling with **pbdPROF**

1. Rebuild **pbdR** packages

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

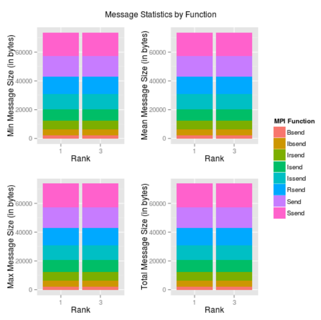
2. Run code

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

3. Analyze results

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

Publication-quality graphs



pbdR Scripts

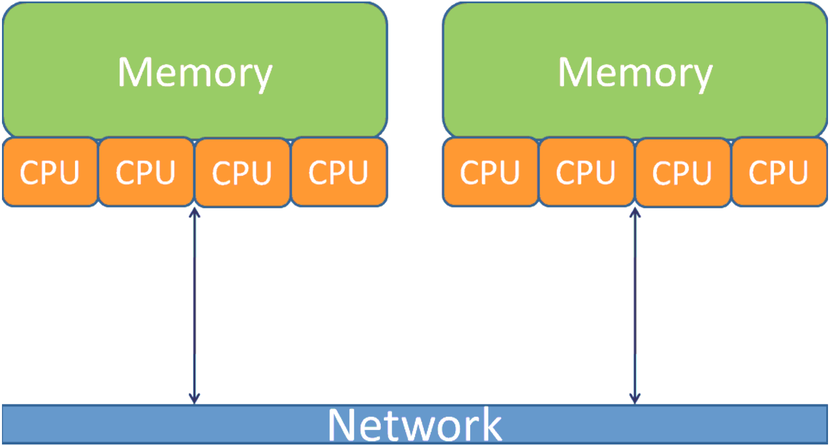
- They're just R scripts.
- Can't run interactively (with more than 1 rank).
- We can use **pbdinline** to get "pretend interactivity".

2 A Hasty Introduction to MPI

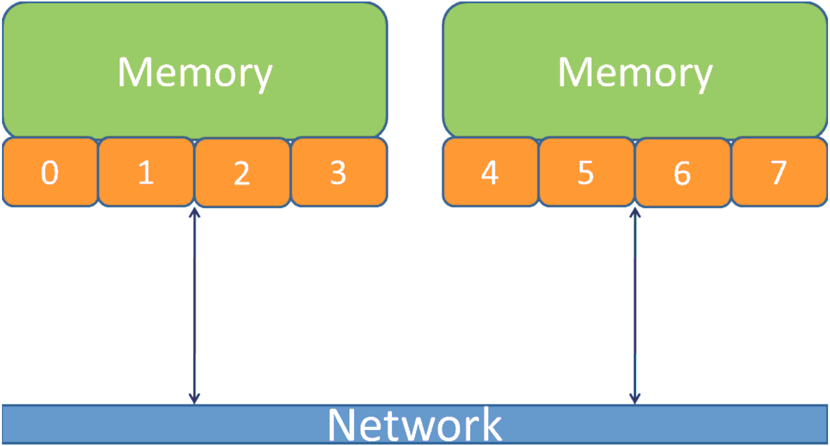
Message Passing Interface (MPI)

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

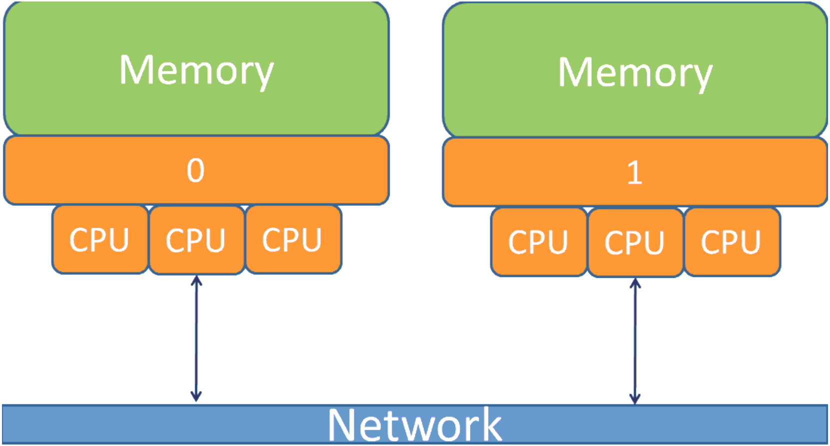
MPI Communicators



MPI Communicators



MPI Communicators



MPI Operations (1 of 2)

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

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

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

Quick Example 1

Rank Query: 1_rank.r

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

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

finalize()
```

Execute this script via:

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

Sample Output:

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

Quick Example 1: pbdinline

```
library(pbdinline)
body <- "
  my.rank <- comm.rank()
  comm.print(my.rank, all.rank=TRUE)
"

pbdRscript(body, cores=2)
```

Quick Example 2

Hello World: 2_hello.r

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

comm.print("Hello, world")

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

finalize()
```

Execute this script via:

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

Sample Output:

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

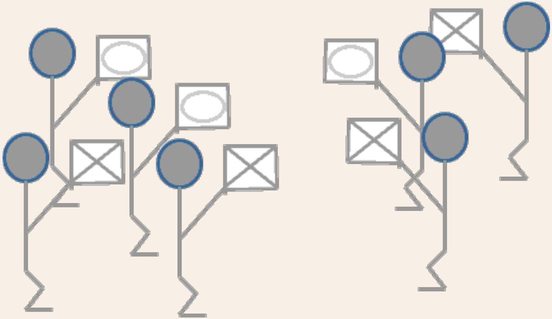
Quick Example 2: pbdinline

```
library(pbdinline)
body <- "
  comm.print("Hello, world")

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

pbdRscript(body, cores=2)
```

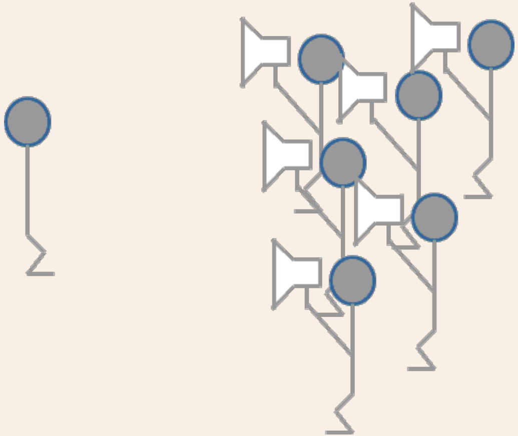
MPI Operations

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

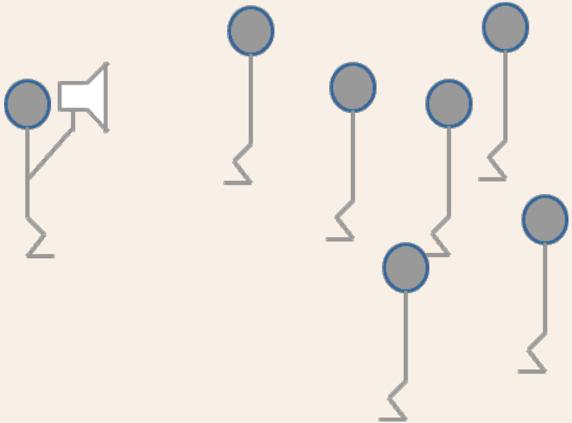
Reductions — Combine results into single result



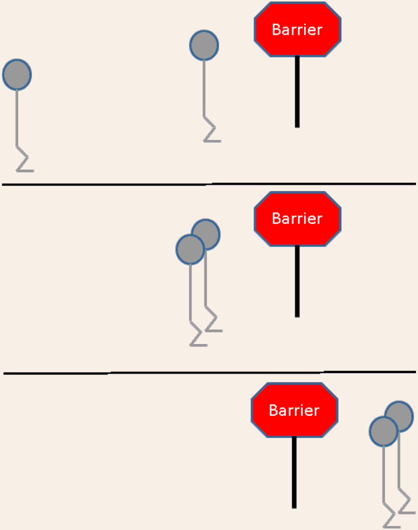
Gather — Many-to-one



Broadcast — One-to-many



Barrier — Synchronization



MPI Operations (2 of 2)

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

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

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

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

Quick Example 3

Reduce and Gather: 3_gt.r

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

comm.set.seed(1234, diff=TRUE)

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

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

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

finalize()
```

Execute this script via:

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

Sample Output:

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

Quick Example 3: pbdinline

```
library(pbdinline)
body <- "
  comm.set.seed(1234, diff=TRUE)

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

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

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

pbdRscript(body, cores=2)
```

Quick Example 4

Broadcast: 4_bcast.r

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

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

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

comm.print(y, rank=1)

finalize()
```

Execute this script via:

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

Sample Output:

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

Quick Example 4: pbdinline

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

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

  comm.print(y, rank=1)
"

pbdRscript(body, cores=2)
```

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()`

3 Distributed Matrices

Distributed Matrices

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

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

ddmatrix: 2-dimensional Block-Cyclic with 6 Processors

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

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

Understanding `ddmatrix`: Local View

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

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

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

Methods for class `ddmatrix`

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

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

Serial Code

```
cov(x)
```

Parallel Code

```
cov(x)
```

ddmatrix Syntax

```
cov.x <- cov(x)
pca <- prcomp(x)
x <- x[, -1]
col.sd <- apply(x, MARGIN=2, FUN=sd)
```

4 Matrix Exponentiation

Exponential Function

Recall from calculus that if $x \in \mathbb{R}$:

$$\exp(x) = \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Matrix Exponentiation

For a square matrix $X_{n\times n}$, we define the matrix exponential:

$$\text{expm}(X) = \frac{1}{1!}X + \frac{1}{2!}X^2 + \frac{1}{3!}X^3 + \dots$$

when $X \neq \mathbf{0}_{n\times n}$; in this case, we take:

$$\text{expm}(\mathbf{0}_{n\times n}) = \mathbf{id}_{n\times n}$$

Computing the Matrix Exponential

- The naive implementation leads to a loss of accuracy for many matrices.
- This problem has been vigorously argued for 30+ years.
- Moler and Van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix*.

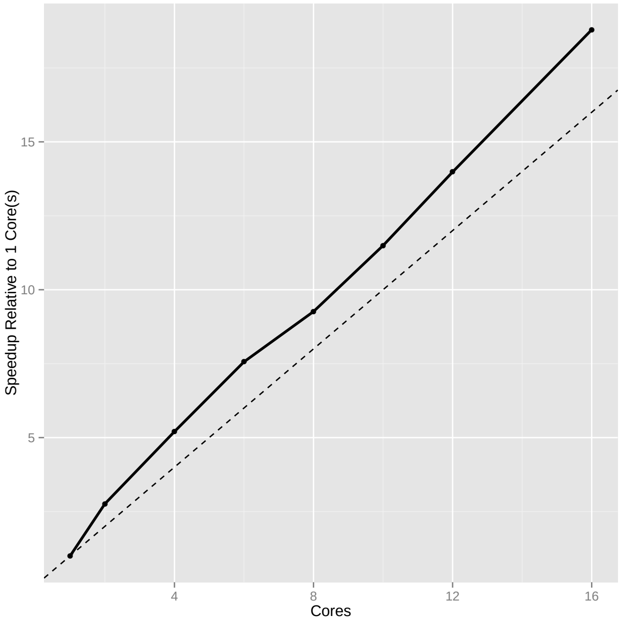
Scaling and Squaring

We use an improvement from Al-Mohy and Higham, *A New Scaling and Squaring Algorithm for the Matrix Exponential*.

```
expm <- function(x)
{
  n <- 2^j
  x <- x/n

  S <- matexp_pade(x)
  S <- matpow_by_squaring(S, n)

  return( S )
}
```

expm()

```r
library(pbdDMAT)

x <- matrix(rnorm(25), 5, 5)
expm(x)

dx <- as.ddmatrix(x)
expm(dx)
```



15
10
5
Speedup Relative to 1 Core(s)
4
8
12
16
Cores

5 Wrapup

Where to Learn More

- The **pbdDEMO** package: http://cran.r-project.org/web/packages/pbdDEMO/index.html
- The **pbdDEMO** vignette, *Speaking Serial R with a Parallel Accent*: http://cran.r-project.org/web/packages/pbdDEMO/vignettes/pbdDEMO-guide.pdf
- Full tutorial at UseR 2014

Thanks for coming!

Questions?