Rdsm—a Tool for Parallel R

Norm Matloff
University of California at Davis

Bay Area R User Group
Intuit, Inc., August 21, 2013

URL for these slides (repeated on final slide):
http://heather.cs.ucdavis.edu/Rdsm
What Is Rdsm?

- tool for parallel R programming
- the only one using the shared-memory paradigm (to be explained)
- advantages:
  - clearer code
  - runs substantially faster in some apps
  - can be used on matrices too large for memory
What Is Rdsm?

- tool for parallel R programming
What Is Rdsm?

• tool for parallel R programming
• the only one using the shared-memory paradigm (to be explained)
What Is Rdsm?

- tool for parallel R programming
- the only one using the shared-memory paradigm (to be explained)
- advantages:
What Is Rdsm?

- tool for parallel R programming
- the only one using the shared-memory paradigm (to be explained)
- advantages:
  - clearer code
What Is Rdsm?

- tool for parallel R programming
- the only one using the shared-memory paradigm (to be explained)
- advantages:
  - clearer code
  - runs substantially faster in some apps
What Is Rdsm?

• tool for parallel R programming
• the only one using the shared-memory paradigm (to be explained)
• advantages:
  • clearer code
  • runs substantially faster in some apps
  • can be used on matrices too large for memory
Rdsm—a Tool for Parallel R

Norm Matloff
University of California at Davis

Rdsm 2.0

- much easier to use
- much faster speed (due to using bigmemory)
- able to store large matrices out of core (due to using bigmemory)
Rdsm—a Tool for Parallel R

Norm Matloff
University of California at Davis

Rdsm 2.0

• much easier to use
• much faster speed (due to using bigmemory)
• able to store large matrices out of core (due to using bigmemory)
Rdsrm 2.0

- much easier to use
Rdsm 2.0

- much easier to use
- much faster speed
Rdsm—A Tool for Parallel R

Norm Matloff
University of California at Davis

Rdsm 2.0

- much easier to use
- much faster speed (due to using bigmemory)
Rdsm 2.0

- much easier to use
- much faster speed (due to using `bigmemory`)
- able to store large matrices out of core
Rdsm 2.0

- much easier to use
- much faster speed (due to using `bigmemory`)
- able to store large matrices out of core (due to using `bigmemory`)

Rdsm—a Tool for Parallel R
Norm Matloff
University of California at Davis
Some Timings to Get Your Attention

• example finds pairwise distance matrix (distances from all rows of X to all rows of Y)
• X, Y both nxn in this example
• use Snow part of parallel library
• times in seconds
Some Timings to Get Your Attention

- example finds pairwise distance matrix
Some Timings to Get Your Attention

- example finds pairwise distance matrix (distances from all rows of $X$ to all rows of $Y$)
Some Timings to Get Your Attention

• example finds pairwise distance matrix (distances from all rows of X to all rows of Y)
• X, Y both nxn in this example
Some Timings to Get Your Attention

- example finds pairwise distance matrix (distances from all rows of X to all rows of Y)
- X, Y both nxn in this example
- use **Snow** part of **parallel** library
Some Timings to Get Your Attention

- example finds pairwise distance matrix (distances from all rows of X to all rows of Y)
- X, Y both nxn in this example
- use Snow part of parallel library
- times in seconds
Distance Matrix Timings

<table>
<thead>
<tr>
<th>n</th>
<th># of cores</th>
<th>Rdsm time</th>
<th>Snow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2</td>
<td>9.264</td>
<td>9.300</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>3.951</td>
<td>4.567</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>2.178</td>
<td>3.196</td>
</tr>
<tr>
<td>2500</td>
<td>24</td>
<td>24.929</td>
<td>39.292</td>
</tr>
</tbody>
</table>
## Distance Matrix Timings

<table>
<thead>
<tr>
<th>n</th>
<th># of cores</th>
<th>Rdsm time</th>
<th>Snow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2</td>
<td>9.264</td>
<td>9.300</td>
</tr>
<tr>
<td>1000</td>
<td>4</td>
<td>3.951</td>
<td>4.567</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>2.178</td>
<td>3.196</td>
</tr>
<tr>
<td>2500</td>
<td>24</td>
<td>24.929</td>
<td>39.292</td>
</tr>
</tbody>
</table>
Shared Memory Programming
Shared Memory Programming

What is shared-memory programming?
Shared Memory Programming

What is shared-memory programming?

- Say \( x \) at process 2, \( y \) at process 5,
Shared Memory Programming

What is shared-memory programming?

- Say \( x \) at process 2, \( y \) at process 5, and want to copy \( x \) to \( y \).
Shared Memory Programming

What is shared-memory programming?

- Say \( x \) at process 2, \( y \) at process 5, and want to copy \( x \) to \( y \).
- \textbf{Rmpi} (message-passing)

\begin{verbatim}
# execute at process 2:
mpi.send.Robj(x, tag=0, dest=5)
# execute at process 5:
y <- mpi.recv.Robj(tag=0, source=2)
\end{verbatim}
What is shared-memory programming?

- Say \( x \) at process 2, \( y \) at process 5, and want to copy \( x \) to \( y \).
- \textbf{Rmpi} (message-passing)
  
  ```
  # execute at process 2:
  mpi.send.Robj(x, tag=0, dest=5)
  # execute at process 5:
  y <- mpi.recv.Robj(tag=0, source=2)
  ```
- \textbf{Rdsm} (shared-memory)
  
  ```
  y <- x
  ```
Shared Memory Programming

What is shared-memory programming?

- Say \(x\) at process 2, \(y\) at process 5, and want to copy \(x\) to \(y\).
- **Rmpi** (message-passing)

```r
# execute at process 2:
mpi.send.Robj(x, tag=0, dest=5)
# execute at process 5:
y <- mpi.recv.Robj(tag=0, source=2)
```

- **Rdsm** (shared-memory)

```r
y <- x
```

- Shared-memory much simpler and clearer!
Rdsm Structure
Rdsm Structure

- Multiple R processes, each running a thread (like C/C++ threads).
Rdsm Structure

- Multiple R processes, each running a thread (like C/C++ threads).
- **Snow** is used to launch the threads.
Rdsm Structure

- Multiple R processes, each running a thread (like C/C++ threads).
- **Snow** is used to launch the threads.
- Multicore machine, data in real shared memory.

Rdsm—a Tool for Parallel R

Norm Matloff
University of California at Davis
Rdsm Structure

- Multiple R processes, each running a thread (like C/C++ threads).
- **Snow** is used to launch the threads.
- Multicore machine, data in real shared memory.
- Single machine, data on disk but appear to be in shared memory.
Rdsm Structure

- Multiple R processes, each running a thread (like C/C++ threads).
- **Snow** is used to launch the threads.
- Multicore machine, data in real shared memory.
- Single machine, data on disk but appear to be in shared memory.
- Future versions will allow use on clusters—multiple machines sharing a common file system.
Simple Rdsm Example
Simple Rdsm Example

```r
library(Rdsm)
c2 <- shmcls(2) # form 2-thread Snow cluster
mgrinit(c2) # initialize Rdsm

mgrmakevar(c2,"m",2,2) # make a shared matrix
m[,] <- 3 # 2x2 matrix of all 3s

# demonstrate shared property:
# at each thread, set id to its Rdsm ID
clusterEvalQ(c2,id <- myinfo$id)
# do m[1,id] <- id^2) at each thread
clusterEvalQ(c2,m[1,id] <- id^2)
# so, thread 1 writes 1 to m[1,1],
# thread 2 writes 4 to m[1,2]
m[,] # prints (1,4)
```
Rdsm Parallel Distance Code
Rdsm Parallel Distance Code

- Piggyback on R’s \texttt{pdist()} function.
Rdsm Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
Rdsm Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
- Each thread calls `pdist()` on its chunk of X, with Y.
Rdsn Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
- Each thread calls `pdist()` on its chunk of X, with Y.
- Each thread puts the result back in the proper portion of the output matrix `dout`.
Rdsm Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
- Each thread calls `pdist()` on its chunk of X, with Y.
- Each thread puts the result back in the proper portion of the output matrix `dout`.
- Remember, `dout` is shared by all the threads, a key point.
Rdsm Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
- Each thread calls `pdist()` on its chunk of X, with Y.
- Each thread puts the result back in the proper portion of the output matrix `dout`.
- Remember, `dout` is shared by all the threads, a key point.
- E.g., quadcore machine, 1000 rows in X.
Rdsm Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
- Each thread calls `pdist()` on its chunk of X, with Y.
- Each thread puts the result back in the proper portion of the output matrix `dout`.
- Remember, `dout` is shared by all the threads, a key point.
- E.g., quadcore machine, 1000 rows in X.
- Thread 2 calls `pdist(x[251:500,],y)`, places result in same rows in output matrix `dout[251:500,y]`. 
Rdsm Parallel Distance Code

- Piggyback on R’s `pdist()` function.
- Give each thread a chunk of the X matrix.
- Each thread calls `pdist()` on its chunk of X, with Y.
- Each thread puts the result back in the proper portion of the output matrix `dout`.
- Remember, `dout` is shared by all the threads, a key point.
- E.g., quadcore machine, 1000 rows in X.
- Thread 2 calls `pdist(x[251:500,],y)`, places result in same rows in output matrix `dout[251:500,y]`.
- Since `dout` is shared, we’re done! (No return value.)
The Rdsm Code
The Rdsm Code

```r
rdsm.pdist <- function(x, y, dout) {
  require(pdist)
  nx <- nrow(x)
  myidxs <- getidxs(nx)
  dout[myidxs,] <- as.matrix(pdist(x[myidxs,], y[,]))
}
```
Another Timing Example:

<table>
<thead>
<tr>
<th># cores</th>
<th>Rdsm time</th>
<th>Snow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4.640</td>
<td>6.398</td>
</tr>
<tr>
<td>16</td>
<td>10.892</td>
<td>18.010</td>
</tr>
<tr>
<td>24</td>
<td>8.778</td>
<td>19.001</td>
</tr>
</tbody>
</table>
Another Timing Example:

Matrix multiply, all matrices n×n.

<table>
<thead>
<tr>
<th>n</th>
<th># cores</th>
<th>Rdsm time</th>
<th>Snow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>8</td>
<td>4.640</td>
<td>6.398</td>
</tr>
<tr>
<td>3000</td>
<td>16</td>
<td>10.892</td>
<td>18.010</td>
</tr>
<tr>
<td>3000</td>
<td>24</td>
<td>8.778</td>
<td>19.001</td>
</tr>
</tbody>
</table>
Other Timing Examples

- Transform an nxn graph adjacency matrix.

\[
\begin{bmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 2 \\
1 & 3 \\
2 & 3 \\
\end{bmatrix}
\]

- \(n\) cores
  - Rdsm time
  - Snow time
  - 2500
  - 2
  - 1.885
  - 12.223

- Snow version probably can be improved, but hard to avoid a lot of time-consuming copying.

- Rdsm’s shared-memory nature avoids the copying.
Other Timing Examples

- Transform an $n \times n$ graph adjacency matrix.

\[
\begin{pmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1 \\
\end{pmatrix} \rightarrow \begin{pmatrix}
1 & 2 \\
1 & 3 \\
2 & 3 \\
\end{pmatrix}
\]
Other Timing Examples

- Transform an nxn graph adjacency matrix.

\[
\begin{pmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1
\end{pmatrix} \rightarrow \begin{pmatrix}
1 & 2 \\
1 & 3 \\
2 & 3
\end{pmatrix}
\]

- Table:

<table>
<thead>
<tr>
<th>n</th>
<th># cores</th>
<th>Rdsm time</th>
<th>Snow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>2</td>
<td>1.885</td>
<td>12.223</td>
</tr>
</tbody>
</table>
Other Timing Examples

- Transform an nxn graph adjacency matrix.

\[
\begin{pmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1
\end{pmatrix} \rightarrow \begin{pmatrix}
1 & 2 \\
1 & 3 \\
2 & 3
\end{pmatrix}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># cores</th>
<th>Rdsm time</th>
<th>Snow time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>2</td>
<td>1.885</td>
<td>12.223</td>
</tr>
</tbody>
</table>

- **Snow** version probably can be improved, but hard to avoid a lot of time-consuming copying.
Other Timing Examples

- Transform an n x n graph adjacency matrix.

\[
\begin{pmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1 \\
\end{pmatrix} \rightarrow \begin{pmatrix}
1 & 2 \\
1 & 3 \\
2 & 3 \\
\end{pmatrix}
\]

- Snow version probably can be improved, but hard to avoid a lot of time-consuming copying.

- Rdsm’s shared-memory nature avoids the copying.
Other Timing Examples (cont’d.)

• bucket sort with sampling

- used multicore part of parallel

<table>
<thead>
<tr>
<th>Rdsm</th>
<th>time multicore</th>
<th>time Rdsm</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000,000</td>
<td>20.594</td>
<td>13.039</td>
</tr>
</tbody>
</table>

- bigmemory fast on matrices but slow on vectors?
Other Timing Examples (cont’d.)

- bucket sort with sampling
Other Timing Examples (cont’d.)

- bucket sort with sampling
- used `multicore` part of `parallel`
Other Timing Examples (cont’d.)

- bucket sort with sampling
- used **multicore** part of **parallel**

<table>
<thead>
<tr>
<th>n</th>
<th># cores</th>
<th>Rdsm time</th>
<th>multicore time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000000</td>
<td>4</td>
<td>20.594</td>
<td>13.039</td>
</tr>
</tbody>
</table>
Other Timing Examples (cont’d.)

- bucket sort with sampling
- used `multicore` part of `parallel`

<table>
<thead>
<tr>
<th>n</th>
<th># cores</th>
<th>Rdsm time</th>
<th>multicore time</th>
</tr>
</thead>
<tbody>
<tr>
<td>500000000</td>
<td>4</td>
<td>20.594</td>
<td>13.039</td>
</tr>
</tbody>
</table>

- `bigmemory` fast on matrices but slow on vectors?
How It Works

• Say using 4 cores, and create shared matrix $m$
• Rdsm launches 4 new R processes; visible e.g. using `ps` on Unix-family systems
• bigmemory redefines the $\left[ \right.$ operator, so that e.g. $m[2,5]$ is rerouted to a location in shared memory
• result: even though the 4 R processes run independently, $m[2,5]$ means the same physical location for all of them, i.e. shared
How It Works

• say using 4 cores, and create shared matrix m
How It Works

- say using 4 cores, and create shared matrix \( m \)
- \textbf{Rdsm} launches 4 new R processes; visible e.g. using \texttt{ps} on Unix-family systems
How It Works

- say using 4 cores, and create shared matrix \( m \)
- \texttt{Rdsm} launches 4 new R processes; visible e.g. using \texttt{ps} on Unix-family systems
- \texttt{bigmemory} redefines the [ operator, so that e.g. \( m[2,5] \) is rerouted to a location in shared memory
How It Works

- say using 4 cores, and create shared matrix \( m \)
- **Rdsm** launches 4 new R processes; visible e.g. using **ps** on Unix-family systems
- **bigmemory** redefines the [ operator, so that e.g. \( m[2,5] \) is rerouted to a location in shared memory
- result: even though the 4 R processes run independently, \( m[2,5] \) means the same physical location for all of them, i.e. shared
Summary

• Rdsm—a Tool for Parallel R
• Norm Matloff
• University of California at Davis
• Available on CRAN
• URL for these slides (repeated on final slide): http://heather.cs.ucdavis.edu/Rdsm
Summary

Rdsm:

• parallel R
• simpler coding; possible performance advantage; works on very large matrices
• available on CRAN

URL for these slides (repeated on final slide):
http://heather.cs.ucdavis.edu/Rdsm
Summary

**Rdsm:**

- parallel R
Summary

Rdsm:

- parallel R
- simpler coding; possible performance advantage; works on very large matrices

Rdsm—a Tool for Parallel R
Norm Matloff
University of California at Davis

URL for these slides (repeated on final slide):
http://heather.cs.ucdavis.edu/Rdsm
Summary

Rdsm:

- parallel R
- simpler coding; possible performance advantage; works on very large matrices
- available on CRAN
- URL for these slides (repeated on final slide): http://heather.cs.ucdavis.edu/Rdsm