Parallel R

Norm Matloff
University of California at Davis

LUGOD
February 17, 2014

URL for these slides (repeated on final slide):
`http://heather.cs.ucdavis.edu/ParallelR.pdf`

What Is R?

What Is R?

- Open source tool for data science.

What Is R?

- Open source tool for data science.
- Open source version of old S (Bell Labs).

What Is R?

- Open source tool for data science.
- Open source version of old S (Bell Labs).
- “We’re not in Statisticsland anymore.”

What Is R?

- Open source tool for data science.
- Open source version of old S (Bell Labs).
- "We're not in Statisticsland anymore."
- Statistically Correct (not all are)...

What Is R?

- Open source tool for data science.
- Open source version of old S (Bell Labs).
- "We're not in Statisticsland anymore."
- Statistically Correct (not all are)... but now used for general data manipulation, and especially graphics

What Is R?

- Open source tool for data science.
- Open source version of old S (Bell Labs).
- "We're not in Statisticsland anymore."
- Statistically Correct (not all are)... but now used for general data manipulation, and especially graphics
- Typically used in interactive mode, like Python.

Some R IDEs

Some R IDEs

- RStudio
  Enormously popular. By JJ Allaire, developer of Cold Fusion long ago.
- ESS—Emacs Speaks Statistics
  For the really hard core R programmers.
- vim-r
  Ditto, but for Vim.
- StatET
  Nice, if you can deal with Eclipse.

Need for Parallel Computation

Need for Parallel Computation

- We're in the era Big Data:

Need for Parallel Computation

- We're in the era Big Data:
    - Large number of data points.

Need for Parallel Computation

- We're in the era Big Data:
  - Large number of data points.
  - Large number of variables.

Need for Parallel Computation

- We're in the era Big Data:
  - Large number of data points.
  - Large number of variables.
- Machine Learning

Need for Parallel Computation

- We're in the era Big Data:
  - Large number of data points.
  - Large number of variables.
- Machine Learning (old nonparametric methods but now rebranded)

Need for Parallel Computation

- We're in the era Big Data:
    - Large number of data points.
    - Large number of variables.
- Machine Learning (old nonparametric methods but now rebranded) tend to be very computationally intensive.

Norm Matloff
University of California at Davis

Obstacles

Obstacles

- R was not designed for parallel computation.

Norm Matloff
University of California at Davis

Obstacles

- R was not designed for parallel computation.
- R is not threaded, probably won't be in the future.

Obstacles

- R was not designed for parallel computation.
- R is not threaded, probably won't be in the future.
- R is a functional language, (mostly) free of side effects,

Obstacles

- R was not designed for parallel computation.
- R is not threaded, probably won't be in the future.
- R is a functional language, (mostly) free of side effects, so assignment of a single matrix element

  ```
  x[622,8888] <- y
  ```

  may cause the entire matrix storage to be reallocated.

Workarounds

Workarounds

All of the below are done, though with some drawbacks.

Workarounds

All of the below are done, though with some drawbacks.

- Implement some fundamental operations, say matrix multiplication in C/OpenMP, then interface to R.

Workarounds

All of the below are done, though with some drawbacks.

- Implement some fundamental operations, say matrix multiplication in C/OpenMP, then interface to R. But still have problems with the anti-side-effects "religion."

Workarounds

All of the below are done, though with some drawbacks.

- Implement some fundamental operations, say matrix multiplication in C/OpenMP, then interface to R. But still have problems with the anti-side-effects "religion."
- Same for GPU.

Workarounds

All of the below are done, though with some drawbacks.

- Implement some fundamental operations, say matrix multiplication in C/OpenMP, then interface to R. But still have problems with the anti-side-effects “religion.”
- Same for GPU.
- Have multiple instantiations of R act in concert.

Workarounds

All of the below are done, though with some drawbacks.

- Implement some fundamental operations, say matrix multiplication in C/OpenMP, then interface to R. But still have problems with the anti-side-effects “religion.”
- Same for GPU.
- Have multiple instantiations of R act in concert. But have overhead from process-to-process copying, especially on clusters.

Workarounds

All of the below are done, though with some drawbacks.

- Implement some fundamental operations, say matrix multiplication in C/OpenMP, then interface to R. But still have problems with the anti-side-effects "religion."
- Same for GPU.
- Have multiple instantiations of R act in concert. But have overhead from process-to-process copying, especially on clusters.

I'll focus on that last approach.

Major World Views

Norm Matloff
University of California at Davis

Major World Views

Major paradigms for general parallel programming:

Major World Views

Major paradigms for general parallel programming:

- **message passing:**

Major World Views

Major paradigms for general parallel programming:

- **message passing:**

```
// copy x (process 3) to y (process 8)
p.3 sends x
p.8 receives
p.8 does y = x
```

  Used on both clusters and multicore.

- **shared-memory:**

Major World Views

Major paradigms for general parallel programming:

- **message passing:**

```
// copy x (process 3) to y (process 8)
p.3 sends x
p.8 receives
p.8 does y = x
```

  Used on both clusters and multicore.

- **shared-memory:**

```
// copy x (process 3) to y (process 8)
y = x
```

  Technically usable only on multicore.

Norm Matloff
University of California at Davis

Extent of Usage

Extent of Usage

- **message passing:**
  - Got a head start, since shared-memory hardware affordable only recently.
  - MPI very popular.
- **shared-memory:**
  - Small, medium multicore, and GPU, now common.
  - OpenMP very popular, misc. (TBB, Cilk++).
  - CUDA is big.

Extent of Usage

- **message passing:**
  - Got a head start, since shared-memory hardware affordable only recently.
  - MPI very popular.
- **shared-memory:**
  - Small, medium multicore, and GPU, now common.
  - OpenMP very popular, misc. (TBB, Cilk++).
  - CUDA is big.

Yet the situation is quite different in parallel R: Message-passing dominates.

Multiprocess R

Norm Matloff
University of California at Davis

Multiprocess R

- **message passing:**

Multiprocess R

- **message passing:**
  - "snow" part of **parallel** (L. Tierney, U. of Iowa)
    Was a contributed package, now part of base R.

Multiprocess R

- **message passing:**
  - "snow" part of **parallel** (L. Tierney, U. of Iowa)
    Was a contributed package, now part of base R.
  - "multicore" part of **parallel** (S. Urbanek, ATT&T)
    Was a contributed package, now part of base R.

Multiprocess R

- **message passing:**
    - "snow" part of **parallel** (L. Tierney, U. of Iowa)
      Was a contributed package, now part of base R.
    - "multicore" part of **parallel** (S. Urbanek, ATT&T)
      Was a contributed package, now part of base R.
    - **Rmpi** (Hao Yu, U. of Western Ontario)
      Contributed.

Multiprocess R

- **message passing:**
  - "snow" part of **parallel** (L. Tierney, U. of Iowa)
    Was a contributed package, now part of base R.
  - "multicore" part of **parallel** (S. Urbanek, ATT&T)
    Was a contributed package, now part of base R.
  - **Rmpi** (Hao Yu, U. of Western Ontario)
    Contributed.
  - **foreach()** (Revolution Analytics)
    Contributed, wrapper to the others above.

Multiprocess R

- **message passing:**
  - "snow" part of **parallel** (L. Tierney, U. of Iowa)
    Was a contributed package, now part of base R.
  - "multicore" part of **parallel** (S. Urbanek, ATT&T)
    Was a contributed package, now part of base R.
  - **Rmpi** (Hao Yu, U. of Western Ontario)
    Contributed.
  - **foreach()** (Revolution Analytics)
    Contributed, wrapper to the others above.
- **shared-memory**

Multiprocess R

- **message passing:**
  - "snow" part of **parallel** (L. Tierney, U. of Iowa)
    Was a contributed package, now part of base R.
  - "multicore" part of **parallel** (S. Urbanek, ATT&T)
    Was a contributed package, now part of base R.
  - **Rmpi** (Hao Yu, U. of Western Ontario)
    Contributed.
  - **foreach()** (Revolution Analytics)
    Contributed, wrapper to the others above.
- **shared-memory**
  - **Rdsm** (NM)

Multiprocess R

- **message passing:**
  - "snow" part of **parallel** (L. Tierney, U. of Iowa)
    Was a contributed package, now part of base R.
  - "multicore" part of **parallel** (S. Urbanek, ATT&T)
    Was a contributed package, now part of base R.
  - **Rmpi** (Hao Yu, U. of Western Ontario)
    Contributed.
  - **foreach()** (Revolution Analytics)
    Contributed, wrapper to the others above.
- **shared-memory**
  - **Rdsm** (NM)
    Contributed.
  - **gputools** (Buckner *et al*, U. of Mich.)
    Contributed.

Sample Application

Sample Application

As a sample application, let's use Mutual Outlinks: Given n Web sites, find the mean number of mutual outlinks over all n(n-1)/2 pairs. (Matrix is coded with 0s and 1s.)

Sample Application

As a sample application, let's use Mutual Outlinks: Given n Web sites, find the mean number of mutual outlinks over all n(n-1)/2 pairs. (Matrix is coded with 0s and 1s.)
Here is the serial code:

Sample Application

As a sample application, let's use Mutual Outlinks: Given n Web sites, find the mean number of mutual outlinks over all n(n-1)/2 pairs. (Matrix is coded with 0s and 1s.)

Here is the serial code:

```
mutoutser <- function(links) {
   nr <- nrow(links); nc <- ncol(links)
   tot = 0
   for (i in 1:(nr-1)) {
      for (j in (i+1):nr) {
         for (k in 1:nc)
            tot <- tot + links[i,k] * links[j,k]
      }
   }
   tot / nr
}
```

Sample Application, cont'd.

Norm Matloff
University of California at Davis

Sample Application, cont'd.

Improvement: 2 loops can be eliminated by noting that they are equivalent to matrix multiplication.

Sample Application, cont'd.

Improvement: 2 loops can be eliminated by noting that they are equivalent to matrix multiplication.

```
for (j in (i+1):nr) {
   for (k in 1:nc)
      tot <- tot + links[i,k] * links[j,k]
}
```

becomes

```
tmp <- links[(i+1):nr,] %*% links[i,]
tot <- tot + sum(tmp)
```

Sample Application, cont'd.

Sample Application, cont'd.

Improved version:

```
mutoutser1<- function(links) {
    nr <- nrow(links)
    nc <- ncol(links)
    tot <- 0
    for (i in 1:(nr-1)) {
        # matrix mult. operator is %*%
        tmp <- links[(i+1):nr,] %*% links[i,]
        tot <- tot + sum(tmp)
    }
    tot / nr
}
```

Norm Matloff
University of California at Davis

Timings

Timings

size	orig.	improved
500x500	106.7s	1.5s

Norm Matloff
University of California at Davis

Timings

size	orig.	improved
500x500	106.7s	1.5s

Wow! Vectorizing really helps.

Timings

size	orig.	improved
500x500	106.7s	1.5s

Wow! Vectorizing really helps.
But even the improved code takes 94.1s for 2000x2000.

Timings

size	orig.	improved
500x500	106.7s	1.5s

Wow! Vectorizing really helps.
But even the improved code takes 94.1s for 2000x2000.
Parallel computation is needed.

How Snow Works

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

Norm Matloff
University of California at Davis

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

- Say have Machines A, B and C, networked.

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

- Say have Machines A, B and C, networked. R is running on all 3.

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

- Say have Machines A, B and C, networked. R is running on all 3.
- "Manager" R process, at A, divvies up the workload, sends chunks to "workers" B, C.

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

- Say have Machines A, B and C, networked. R is running on all 3.
- "Manager" R process, at A, divvies up the workload, sends chunks to "workers" B, C.
- B, C work on their chunks, send results back to A.

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

- Say have Machines A, B and C, networked. R is running on all 3.
- "Manager" R process, at A, divvies up the workload, sends chunks to "workers" B, C.
- B, C work on their chunks, send results back to A.
- R process A receives, and combines the results into the final answer.

Norm Matloff
University of California at Davis

How Snow Works

The **snow** contributed package is now part of base R, in the **parallel** package.

- Say have Machines A, B and C, networked. R is running on all 3.
- "Manager" R process, at A, divvies up the workload, sends chunks to "workers" B, C.
- B, C work on their chunks, send results back to A.
- R process A receives, and combines the results into the final answer.

Communication between R processes done by sockets or other.

Norm Matloff
University of California at Davis

Mut. Outs. in Snow

Mut. Outs. in Snow

```
doichunk <- function(ichunk) {
   tot <- 0
   nr <- nrow(lnks)
   for (i in ichunk) {
      tmp <- lnks[(i+1):nr,] %*% lnks[i,]
      tot <- tot + sum(tmp)
   }
   tot
}
mutoutpar <- function(cls) {
   require(parallel)
   nr <- nrow(lnks)
   clusterExport(cls,"lnks")
   ichunks <- 1:(nr-1)
   tots <- clusterApply(cls,ichunks,doichunk)
   Reduce(sum,tots) / nr
}
```

Norm Matloff
University of California at Davis

Timings

Timings

Timing, dual-core, machine, but hyperthreaded.

size	improved.	2 wrkrs.	4 wrkrs.
2000x2000	94.5s	80.3s	70.1s

Timings

Timing, dual-core, machine, but hyperthreaded.

size	improved.	2 wrkrs.	4 wrkrs.
2000x2000	94.5s	80.3s	70.1s

Get improvement, though not the theoretical 2X and 4X.

Norm Matloff
University of California at Davis

Norm Matloff
University of California at Davis

Overhead in Snow

Overhead in Snow

- Data copied from manager to workers at beginning of run.

Overhead in Snow

- Data copied from manager to workers at beginning of run.
- Data copied from workers to manager at end of run.

Overhead in Snow

- Data copied from manager to workers at beginning of run.
- Data copied from workers to manager at end of run.
- More copying from manager to manager at end of run; see calls to **Reduce()** above.

How Multicore Works

[1]Works on Unix-family systems only. Also, **snow** now includes this kind of option

How Multicore Works

The **multicore** contributed package is now part of base R, in the **parallel** package.

[1] Works on Unix-family systems only. Also, **snow** now includes this kind of option

How Multicore Works

The **multicore** contributed package is now part of base R, in the **parallel** package.

- API, operation similar to **snow**.

[1]Works on Unix-family systems only. Also, **snow** now includes this kind of option

How Multicore Works

The **multicore** contributed package is now part of base R, in the **parallel** package.

- API, operation similar to **snow**.
- Should be somewhat faster than **snow**,

[1]Works on Unix-family systems only. Also, **snow** now includes this kind of option

How Multicore Works

The **multicore** contributed package is now part of base R, in the **parallel** package.

- API, operation similar to **snow**.
- Should be somewhat faster than **snow**, as it uses **fork()** on the original (manager) R process[1]—no copying data at the beginning.

[1] Works on Unix-family systems only. Also, **snow** now includes this kind of option

How Multicore Works

The **multicore** contributed package is now part of base R, in the **parallel** package.

- API, operation similar to **snow**.
- Should be somewhat faster than **snow**, as it uses **fork()** on the original (manager) R process[1]—no copying data at the beginning.
- But has the same copying delays at the end.

[1] Works on Unix-family systems only. Also, **snow** now includes this kind of option

How Multicore Works

The **multicore** contributed package is now part of base R, in the **parallel** package.

- API, operation similar to **snow**.
- Should be somewhat faster than **snow**, as it uses **fork()** on the original (manager) R process[1]—no copying data at the beginning.
- But has the same copying delays at the end.

[1]Works on Unix-family systems only. Also, **snow** now includes this kind of option

Norm Matloff
University of California at Davis

The foreach() Package

[2] And add **%dopar%**.

The foreach() Package

- Probably the most popular type of parallel R currently.

[2] And add **%dopar%**.

The foreach() Package

- Probably the most popular type of parallel R currently.
- Actually just a wrapper to **snow**, **multicore** etc.

[2] And add **%dopar%**.

The foreach() Package

- Probably the most popular type of parallel R currently.
- Actually just a wrapper to **snow**, **multicore** etc.
- Major attraction:

---

[2] And add **%dopar%**.

The foreach() Package

- Probably the most popular type of parallel R currently.
- Actually just a wrapper to **snow**, **multicore** etc.
- Major attraction: Just replace **for()** in your serial code with **foreach()**![2]

[2] And add **%dopar%**.

The foreach() Package

- Probably the most popular type of parallel R currently.
- Actually just a wrapper to **snow**, **multicore** etc.
- Major attraction: Just replace **for()** in your serial code with **foreach()**![2]
  E.g. Mutual Outlinks:

```
foreach(i = 1:(nr-1)) %dopar% {
   for (j in (i+1):nr) {
      for (k in 1:nc)
         tot <- tot + links[i,k] * links
   }
}
```

[2] And add **%dopar%**.

The foreach() Package

- Probably the most popular type of parallel R currently.
- Actually just a wrapper to **snow**, **multicore** etc.
- Major attraction: Just replace **for()** in your serial code with **foreach()**![2]
  E.g. Mutual Outlinks:

```
foreach(i = 1:(nr-1)) %dopar% {
   for (j in (i+1):nr) {
      for (k in 1:nc)
         tot <- tot + links[i,k] * links
   }
}
```

- But that "attraction" is a "fatal attraction"...

[2] And add **%dopar%**.

Norm Matloff
University of California at Davis

More on foreach()

More on foreach()

- Simply replacing **foreach()** can really rob your code of speed.

More on foreach()

- Simply replacing **foreach()** can really rob your code of speed.
- E.g. Mutual Outlinks.

More on foreach()

- Simply replacing **foreach()** can really rob your code of speed.
- E.g. Mutual Outlinks. The original serial code did NOT take advantage of matrix multiplication,

More on foreach()

- Simply replacing **foreach()** can really rob your code of speed.
- E.g. Mutual Outlinks. The original serial code did NOT take advantage of matrix multiplication, so a naive use of **foreach()** can cause a substantial slowdown:

More on foreach()

- Simply replacing **foreach()** can really rob your code of speed.
- E.g. Mutual Outlinks. The original serial code did NOT take advantage of matrix multiplication, so a naive use of **foreach()** can cause a substantial slowdown:

size	# wrkrs.	**foreach()**	**snow**
500x500	2	17.7s	11.3s
500x500	4	13.6s	6.0s
500x500	8	7.4s	3.4s

Of course, you can parameterize your **for()** loop to use chunking, but this weakens the appeal of being able to simply change one line of one's serial code.

Rmpi

Rmpi

- Provides R interfaces to most MPI functions,

Rmpi

- Provides R interfaces to most MPI functions, plus some new ones specific to R.

Rmpi

- Provides R interfaces to most MPI functions, plus some new ones specific to R.
- Very versatile.

Rmpi

- Provides R interfaces to most MPI functions, plus some new ones specific to R.
- Very versatile.
- Can be a (big) pain to configure.

Rdsm

Rdsm

- Shared-memory.

Rdsm

- Shared-memory.
- Add threads to R programming!

Rdsm

- Shared-memory.
- Add threads to R programming!
- Builds on my old parallel Perl package, PerlDSM.

Rdsm

- Shared-memory.
- Add threads to R programming!
- Builds on my old parallel Perl package, PerlDSM.

Rdsm Shared-Memory

[3] Remember, R is a functional language. Even array read/write are functions.

Rdsm Shared-Memory

- R's array-access function **"["()**[3] is overloaded, with the access being rerouted.

[3] Remember, R is a functional language. Even array read/write are functions.

Rdsm Shared-Memory

- R's array-access function **"["()**[3] is overloaded, with the access being rerouted.
- In **Rdsm 1.0**, array access was routed to a server.

[3]Remember, R is a functional language. Even array read/write are functions.

Rdsm Shared-Memory

- R's array-access function **"["()**[3] is overloaded, with the access being rerouted.
- In **Rdsm 1.0**, array access was routed to a server.
- In **Rdsm 2.0**, array access is built on top of the R package **bigmemory**.

[3] Remember, R is a functional language. Even array read/write are functions.

Norm Matloff
University of
California at
Davis

Rdsm Shared-Memory (cont'd.)

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.
- The **bigmemory** package is not a parallel programming system.

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.
- The **bigmemory** package is not a parallel programming system.
- **Rdsm** adds parallel programming structure on top of **bigmemory**.

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.
- The **bigmemory** package is not a parallel programming system.
- **Rdsm** adds parallel programming structure on top of **bigmemory**.
- R's **bigmemory** is perfect for **Rdsm**;

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.
- The **bigmemory** package is not a parallel programming system.
- **Rdsm** adds parallel programming structure on top of **bigmemory**.
- R's **bigmemory** is perfect for **Rdsm**; it creates physically shared memory, using Unix **shmget()** etc.

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.
- The **bigmemory** package is not a parallel programming system.
- **Rdsm** adds parallel programming structure on top of **bigmemory**.
- R's **bigmemory** is perfect for **Rdsm**; it creates physically shared memory, using Unix **shmget()** etc.
- Still have multiple R processes, as with **snow** etc., but they all read/write the same physical memory locations.

Rdsm Shared-Memory (cont'd.)

- Goals of **bigmemory**: larger address space and ability to write to arrays without reallocation.
- The **bigmemory** package is not a parallel programming system.
- **Rdsm** adds parallel programming structure on top of **bigmemory**.
- R's **bigmemory** is perfect for **Rdsm**; it creates physically shared memory, using Unix **shmget()** etc.
- Still have multiple R processes, as with **snow** etc., but they all read/write the same physical memory locations.
- **snow** is used to launch the threads.

Norm Matloff
University of California at Davis

Some Rdsm APIs

Some Rdsm APIs

```
mgrinit():   initialize system
mgrmakevar():   create a shared variable
mgrmakelock():   create a lock
makebarr():   create a barrier
etc.
```

“Hello World” in Rdsm

"Hello World" in Rdsm

- Actually, matrix multiplication, the "Hello World" of the parallel processing community. :-)

"Hello World" in Rdsm

- Actually, matrix multiplication, the "Hello World" of the parallel processing community. :-)

```
#code executed by each thread:
mmul <- function(u,v,w) {
   # decide which rows of u this thread
   # will work on
   myidxs <- splitIndices(nrow(u),
      myinfo$nwrkrs)[[myinfo$id]]
   # multiply this thread's part of u with
   # v, placing the product in the corresp.
   # part of w
   w[myidxs,] <- u[myidxs,] \%*\% v[,]
}
```

Launching the Threads

Launching the Threads

```
# the cluster*() functions are from Snow
# send mmul() to the threads
clusterExport(c2,"mmul")
# run the threads
clusterEvalQ(c2,mmul(a,b,c))
c[,]   # check results
```

Rdsm Can Bring a Substantial Performance Improvement

Rdsm Can Bring a Substantial Performance Improvement

snow vs. **Rdsm**, nxn matrix multiply timings:

Rdsm Can Bring a Substantial Performance Improvement

snow vs. **Rdsm**, nxn matrix multiply timings:

n	# cores	**Rdsm**	**Snow**
2000	8	4.640	6.398
3000	16	10.892	18.010
3000	24	8.778	19.001

Rdsm Can Bring a Substantial Performance Improvement

snow vs. **Rdsm**, nxn matrix multiply timings:

n	# cores	**Rdsm**	**Snow**
2000	8	4.640	6.398
3000	16	10.892	18.010
3000	24	8.778	19.001

The problem with **snow** (and **multicore**):

Rdsm Can Bring a Substantial Performance Improvement

snow vs. **Rdsm**, nxn matrix multiply timings:

n	# cores	**Rdsm**	**Snow**
2000	8	4.640	6.398
3000	16	10.892	18.010
3000	24	8.778	19.001

The problem with **snow** (and **multicore**):
Too much data copying!

Norm Matloff
University of California at Davis

Debugging

Debugging

- R includes some terminal-based primitive debugging tools (and IDEs include some nicer ones).

Norm Matloff
University of California at Davis

Debugging

- R includes some terminal-based primitive debugging tools (and IDEs include some nicer ones).
- However, **snow**, **multicore** etc. **don't have a terminal!**. :-(

Debugging

- R includes some terminal-based primitive debugging tools (and IDEs include some nicer ones).
- However, **snow**, **multicore** etc. **don't have a terminal!**. :-(
- In **snow**, there at least is a "manual" mode, in which one can set up terminals in a very kludgy manner.

Debugging

- R includes some terminal-based primitive debugging tools (and IDEs include some nicer ones).
- However, **snow**, **multicore** etc. **don't have a terminal!**. :-(
- In **snow**, there at least is a "manual" mode, in which one can set up terminals in a very kludgy manner.
- I have developed my own debugging tool for **snow**, automating and hiding the kludge.

Debugging

- R includes some terminal-based primitive debugging tools (and IDEs include some nicer ones).
- However, **snow**, **multicore** etc. **don't have a terminal!**. :-(
- In **snow**, there at least is a "manual" mode, in which one can set up terminals in a very kludgy manner.
- I have developed my own debugging tool for **snow**, automating and hiding the kludge. Since **Rdsm** is launched by **snow**, my debugger works for **Rdsm** too.

Norm Matloff
University of California at Davis

Debugging

- R includes some terminal-based primitive debugging tools (and IDEs include some nicer ones).
- However, **snow**, **multicore** etc. **don't have a terminal!**. :-(
- In **snow**, there at least is a "manual" mode, in which one can set up terminals in a very kludgy manner.
- I have developed my own debugging tool for **snow**, automating and hiding the kludge. Since **Rdsm** is launched by **snow**, my debugger works for **Rdsm** too.
- As to **multicore**, the situation looks grim.

What About GPU?

What About GPU?

- R's **gputools** package offers some functions, mainly for linear algebra operations.

What About GPU?

- R's **gputools** package offers some functions, mainly for linear algebra operations.
- NVIDIA's Thrust package offers a number of C++ routines for various parallel ops.

What About GPU?

- R's **gputools** package offers some functions, mainly for linear algebra operations.
- NVIDIA's Thrust package offers a number of C++ routines for various parallel ops.
  - Use chooses "back end," either GPU, OpenMP or TBB.

What About GPU?

- R's **gputools** package offers some functions, mainly for linear algebra operations.
- NVIDIA's Thrust package offers a number of C++ routines for various parallel ops.
  - Use chooses "back end," either GPU, OpenMP or TBB.
  - So, your same code can work either on GPU or multicore systems!

What About GPU?

- R's **gputools** package offers some functions, mainly for linear algebra operations.
- NVIDIA's Thrust package offers a number of C++ routines for various parallel ops.
  - Use chooses "back end," either GPU, OpenMP or TBB.
  - So, your same code can work either on GPU or multicore systems!
  - I have developed an R interface to some Thrust-based functions, named **Rth**.

URLs

URLs

- CRAN, for **Rdsm 2.0**, **foreach()**:
  `cran.us.r-project.org`
- **Rdsm 2.1**:
  `heather.cs.ucdavis.edu/Rdsm_2.1.1.tar.gz`
- my **snow**/**Rdsm** debugging tool:
  `heather.cs.ucdavis.edu/DebugSnow_1.0.0.tar.gz`
- **Rth**:
  `heather.cs.ucdavis.edu/~matloff/rth.html`
- rough draft of the first 1/2 of my forthcoming book, *Parallel Computation for Data Science*:
  `heather.cs.ucdavis.edu/paralleldatasci.pdf`
- these slides:
  `heather.cs.ucdavis.edu/ParallelR.pdf`