Statistical Cinderella: Parallel Computation for the Rest of Us

Norm Matloff
University of California at Davis

R/Finance 2018
Chicago, IL, USA, 1 June, 2018

These slides will be available at
`http://heather.cs.ucdavis.edu/rifinance2018.pdf`

Disclaimer

Disclaimer

- "Everyone has opinions."
- I'll present mine.
- Dissent is encouraged. :-)

The Drivers and Their Result

The Drivers and Their Result

- Parallel hardware for the masses:

Norm Matloff
University of California at Davis

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data

The Drivers and Their Result

- Parallel hardware for the masses:
    - 4 cores standard, 16 not too expensive
    - GPUs
    - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
    - Whatever that is.

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
  - Whatever that is.

Result: Users believe,

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
  - Whatever that is.

Result: Users believe,

> *"I've got the hardware and I've got the data need — so I should be all set to do parallel computation in R on the data."*

The Drivers and Their Result

- Parallel hardware for the masses:
  - 4 cores standard, 16 not too expensive
  - GPUs
  - Intel Xeon Phi, $\approx$ 60 cores (!), coprocessor, as low as a few hundred dollars
- Big Data
  - Whatever that is.

Result: Users believe,

> *"I've got the hardware and I've got the data need — so I should be all set to do parallel computation in R on the data."*

But this "rule" is "honored in the breach," as the Brits say.

Not So Simple

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.

Not So Simple

- “Embarrassingly parallel” (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining.

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:
    - Contention for memory/network.

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:
    - Contention for memory/network.
    - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.

Not So Simple

- “Embarrassingly parallel” (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).
  - Contention for I/O ports.

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).
  - Contention for I/O ports.
  - OS/R limits on number of sockets (network connections).

Not So Simple

- "Embarrassingly parallel" (EP) vs. non-EP algorithms.
- EP: Problem can be easily broken down in independent tasks, with easy combining. Embarrassing is good — but not common enough.
- Overhead issues:
  - Contention for memory/network.
  - Bandwidth limits — CPU/memory, CPU/network, CPU/GPU.
  - Cache coherency problems (inconsistent caches in multicore systems).
  - Contention for I/O ports.
  - OS/R limits on number of sockets (network connections).
  - Serialization.

Wish List

Wish List

- Ability to run on various types of hardware — from R.

Wish List

- Ability to run on various types of hardware — from R.
- Ease of use for the non-cognoscenti.

Wish List

- Ability to run on various types of hardware — from R.
- Ease of use for the non-cognoscenti.
- Parameters to tweak for the experts or the daring.

Outline of My Remarks

Outline of My Remarks

- Overview of existing parallel computation options for R users.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
    - Level in terms of abstraction, i.e. high-level constructs.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
  - Level in terms of abstraction, i.e. high-level constructs.
  - Level in terms of tech sophistication needed.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
  - Level in terms of abstraction, i.e. high-level constructs.
  - Level in terms of tech sophistication needed.

"Help, I'm in over my head here!" – a prominent R developer, entering the parallel comp. world.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
  - Level in terms of abstraction, i.e. high-level constructs.
  - Level in terms of tech sophistication needed.

> *"Help, I'm in over my head here!" – a prominent R developer, entering the parallel comp. world.*

- "Cinderellas": Many users are being overlooked.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
    - Level in terms of abstraction, i.e. high-level constructs.
    - Level in terms of tech sophistication needed.

> *"Help, I'm in over my head here!" – a prominent R developer, entering the parallel comp. world.*

- "Cinderellas": Many users are being overlooked.
    - Not enough automatic, tranparent parallelism.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
  - Level in terms of abstraction, i.e. high-level constructs.
  - Level in terms of tech sophistication needed.

"Help, I'm in over my head here!" – a prominent R developer, entering the parallel comp. world.

- "Cinderellas": Many users are being overlooked.
  - Not enough automatic, tranparent parallelism.
  - Not enough for quants, e.g. for time series methods.

Outline of My Remarks

- Overview of existing parallel computation options for R users.
  - Level in terms of abstraction, i.e. high-level constructs.
  - Level in terms of tech sophistication needed.

"Help, I'm in over my head here!" – a prominent R developer, entering the parallel comp. world.

- "Cinderellas": Many users are being overlooked.
  - Not enough automatic, tranparent parallelism.
  - Not enough for quants, e.g. for time series methods.
- Well then, what can be done?

Available Software

Available Software

software	abstr. level	handle non-EP	sophis. level
C++/OpenMP	low	very good	very high
C++/GPU	low	poor	super high
RcppPar. pkg	low	good	very high
parallel pkg	medium	medium	medium
Rdsm pkg	low	good	high
Spark/R pkgs	medium	poor	high
Rmpi pkg	low	good	high
foreach pkg	medium	poor	medium
partools pkg	medium	good	high medium
future pkg	medium	medium	high medium

(OpenMP: standard library for parallelizing on multicore)

The Takeaways

Norm Matloff
University of California at Davis

The Takeaways

- R has an impressive array of parallel software tools available.

The Takeaways

- R has an impressive array of parallel software tools available. **Better than Python!**

The Takeaways

- R has an impressive array of parallel software tools available. **Better than Python!**
- However, all of those tools require a fair amount of programming sophistication to use.

Speciality Packages with Transparent Parallelism

Speciality Packages with Transparent Parallelism

- Using OpenMP, e.g. **xgboost**, **recosystem**.

Speciality Packages with Transparent Parallelism

- Using OpenMP, e.g. **xgboost**, **recosystem**.
- Using GPU, e.g. **gmatrix** (not active?).

Speciality Packages with Transparent Parallelism

- Using OpenMP, e.g. **xgboost**, **recosystem**.
- Using GPU, e.g. **gmatrix** (not active?).
- Even though transparent to the user in principle, **may still need expertise in hardware/systems** to make it run well.

Speciality Packages with Transparent Parallelism

- Using OpenMP, e.g. **xgboost**, **recosystem**.
- Using GPU, e.g. **gmatrix** (not active?).
- Even though transparent to the user in principle, **may still need expertise in hardware/systems** to make it run well. E.g. choice of number of threads, memory capacity issues.

In Other Words...

In Other Words...

- We need to **FACE FACTS**.

In Other Words...

- We need to **FACE FACTS**.
- The days in which data scientists could rely on "black boxes" are GONE.

In Other Words...

- We need to **FACE FACTS**.
- The days in which data scientists could rely on "black boxes" are GONE.
- One needs to have at least some knowledge of the innards:

In Other Words...

- We need to **FACE FACTS**.
- The days in which data scientists could rely on “black boxes” are GONE.
- One needs to have at least some knowledge of the innards:
    - Machine Learning tuning parameters — defaults underfit, naive grid search selection overfits.

In Other Words...

- We need to **FACE FACTS**.
- The days in which data scientists could rely on “black boxes” are GONE.
- One needs to have at least some knowledge of the innards:
  - Machine Learning tuning parameters — defaults underfit, naive grid search selection overfits.
  - For effective parallel computation, one must be adept at coding and at software “tuning parameters,” e.g. number of threads.

In Other Words...

- We need to **FACE FACTS**.
- The days in which data scientists could rely on "black boxes" are GONE.
- One needs to have at least some knowledge of the innards:
  - Machine Learning tuning parameters — defaults underfit, naive grid search selection overfits.
  - For effective parallel computation, one must be adept at coding and at software "tuning parameters," e.g. number of threads.
- Little or no hope for good automatic parallelism.

Parallel Computation of Time Series Analyses

Parallel Computation of Time Series Analyses

Now let's turn to time series.

Parallel Computation of Time Series Analyses

Now let's turn to time series. (Disclaimer: I am not an expert in time series.)

Parallel Computation of Time Series Analyses

Now let's turn to time series. (Disclaimer: I am not an expert in time series.)

- Some parallel methods have been developed.

Parallel Computation of Time Series Analyses

Now let's turn to time series. (Disclaimer: I am not an expert in time series.)

- Some parallel methods have been developed.
- E.g., if large matrices are involved (say models with long memory), one can use OpenMP to parallelize matrix computations

Parallel Computation of Time Series Analyses

Now let's turn to time series. (Disclaimer: I am not an expert in time series.)

- Some parallel methods have been developed.
- E.g., if large matrices are involved (say models with long memory), one can use OpenMP to parallelize matrix computations
- In some cases one can find a clever way to parallelize a specific algorithm (F. Belletti, arXiv, 2015).

Parallel Computation of Time Series Analyses

Now let's turn to time series. (Disclaimer: I am not an expert in time series.)

- Some parallel methods have been developed.
- E.g., if large matrices are involved (say models with long memory), one can use OpenMP to parallelize matrix computations
- In some cases one can find a clever way to parallelize a specific algorithm (F. Belletti, arXiv, 2015).
- But it's much harder than for i.i.d. models.

Possible Obstacles

Possible Obstacles

- Matrix addition/multiplication is EP, but inversion is not.

Possible Obstacles

- Matrix addition/multiplication is EP, but inversion is not.
- Parallelization based on math structures difficult to show asymptotic validity.

Possible Obstacles

- Matrix addition/multiplication is EP, but inversion is not.
- Parallelization based on math structures difficult to show asymptotic validity.
- Breaking t.s. data into chunks might not be EP, due to boundary effects.

Possible Obstacles

- Matrix addition/multiplication is EP, but inversion is not.
- Parallelization based on math structures difficult to show asymptotic validity.
- Breaking t.s. data into chunks might not be EP, due to boundary effects. E.g. computing number of consecutive periods in which value is above a threshold — could span two chunks, or even more.

Possible Obstacles

- Matrix addition/multiplication is EP, but inversion is not.
- Parallelization based on math structures difficult to show asymptotic validity.
- Breaking t.s. data into chunks might not be EP, due to boundary effects. E.g. computing number of consecutive periods in which value is above a threshold — could span two chunks, or even more.
- Hyndman's Rule: Any time series model eventually starts to go bad after very long lengths.

Our partools Package

Norm Matloff
University of California at Davis

Our partools Package

- On CRAN, but go to github.com/matloff for the latest version.

Our partools Package

- On CRAN, but go to github.com/matloff for the latest version.
- Large variety (78+) of functions for parallel **data manipulation** and **computation**.

Our partools Package

- On CRAN, but go to github.com/matloff for the latest version.
- Large variety (78+) of functions for parallel **data manipulation** and **computation**.
- Some functions do a lot, some just a little. The latter can be combined into powerful tools, as with Unix/Linux/Mac scripting.

Our partools Package

- On CRAN, but go to github.com/matloff for the latest version.
- Large variety (78+) of functions for parallel **data manipulation** and **computation**.
- Some functions do a lot, some just a little. The latter can be combined into powerful tools, as with Unix/Linux/Mac scripting.
- Built on top of **parallel** pkg., plus our own MPI-like internal system.

Themes

Themes

- "Leave It There" (LIT) theme: Keep data distributed as long as possible throughout an analysis session, to avoid costly communications delays.

Themes

- "Leave It There" (LIT) theme: Keep data distributed as long as possible throughout an analysis session, to avoid costly communications delays. Borrows distrib. object approach from Hadoop/Spark but much more flexible.

Themes

- "Leave It There" (LIT) theme: Keep data distributed as long as possible throughout an analysis session, to avoid costly communications delays. Borrows distrib. object approach from Hadoop/Spark but much more flexible.
- "Software Alchemy" — convert non-EP to stat. equivalent EP, thus easy parallelization.

Possible Obstacles

Possible Obstacles

Can we extend **partools** to time series applications?

Possible Obstacles

Can we extend **partools** to time series applications? Must overcome:

Possible Obstacles

Can we extend **partools** to time series applications? Must overcome:

- Boundary effects problems.
- SA predicated on i.i.d.

Norm Matloff
University of California at Davis

Sample partools Session

Sample partools Session

- Wikipedia page-access data, Kaggle, 145063 time series of length 550.

Sample partools Session

- Wikipedia page-access data, Kaggle, 145063 time series of length 550.
- Say we wish to run **arma()** for each page. Each is quick, but 145K of them takes some time. Say we are interested only in **ar1**.

Sample partools Session

- Wikipedia page-access data, Kaggle, 145063 time series of length 550.
- Say we wish to run **arma()** for each page. Each is quick, but 145K of them takes some time. Say we are interested only in **ar1**.
- Afterward, we will perform various other operations.

Sample partools Session

- Wikipedia page-access data, Kaggle, 145063 time series of length 550.
- Say we wish to run **arma()** for each page. Each is quick, but 145K of them takes some time. Say we are interested only in **ar1**.
- Afterward, we will perform various other operations.
- By LIT Principle, first distribute the data to the workers, then avoid collecting it back to the manager node if possible.

Serial Version

Serial Version

```
wd <- as.matrix(read.csv('train_1.csv'))
wdc <- wd[complete.cases(wd),]
armac <- function(x)
   {z <- NA; try(z <- arma(x)$coef[1]); z)
system.time(z <- apply(wdc,1,armac))
# 624.452    0.164  624.648
# find the ones with weak correlation
# for further analysis
wdlt05 <- wdc[z < 0.5,]
# various further ops (not shown)
```

Same But with partools

Same But with partools

The plan:

Same But with partools

The plan:

- Distribute the data and LIT. Work on it solely in distributed form as much as possible.

Same But with partools

The plan:

- Distribute the data and LIT. Work on it solely in distributed form as much as possible.
- Distrib. by calling **partools:::distribsplit()**, then later save using **partools:::filesave()**.

Same But with partools

The plan:

- Distribute the data and LIT. Work on it solely in distributed form as much as possible.
- Distrib. by calling **partools:::distribsplit()**, then later save using **partools:::filesave()**.
- The chunks all have the same name, in this case **wdc**. The manager then issues commands via **clusterEvalQ()**, the same command to each worker.

Same But with partools

The plan:

- Distribute the data and LIT. Work on it solely in distributed form as much as possible.
- Distrib. by calling **partools:::distribsplit()**, then later save using **partools:::filesave()**.
- The chunks all have the same name, in this case **wdc**. The manager then issues commands via **clusterEvalQ()**, the same command to each worker.
- At end of session, save to **partools** distributed file, **so don't need to redistribute next time.**

Parallel Version

Parallel Version

```
cls ← makeCluster(4)  # 'parallel' cluster
setclsinfo(cls)  # init 'partools'
distribsplit(cls,'wdc')  # distrib. to workers
clusterEvalQ(cls,library(tseries))
clusterExport(cls,'armac')
system.time(clusterEvalQ(cls,
   ar1 ← apply(wdc,1,armac)))
#    0.024    0.000  180.653
clusterEvalQ(cls,wdlt05 ← wdc[ar1 < 0.5,])  # LIT!
# various further ops (not shown)
# now save, in wdc.1, wdc.2,...
filesave(cls,'wdc','wdc',1,',')
```

Parallel Version

```
cls ← makeCluster(4)  # 'parallel' cluster
setclsinfo(cls)  # init 'partools'
distribsplit(cls,'wdc')  # distrib. to workers
clusterEvalQ(cls,library(tseries))
clusterExport(cls,'armac')
system.time(clusterEvalQ(cls,
   ar1 ← apply(wdc,1,armac)))
#   0.024   0.000 180.653
clusterEvalQ(cls,wdlt05 ← wdc[ar1 < 0.5,])  # LIT!
# various further ops (not shown)
# now save, in wdc.1, wdc.2,...
filesave(cls,'wdc','wdc',1,',')
```

The **one-time** overhead of distributing the data will continue to pay off in further analyses.

More LIT Helpers

More LIT Helpers

- Can distribute the file directly, using **partools:::filesplit()**.
- The functions **fileread()** and **filesave()** automatically add a suffix to the name for chunk number, e.g. **wdc.1**.
- If do need to "undistribute," **distribcat()** will do so, adding the proper header.
- Functions such as **dwhich.min()** treat a distributed data frame as a virtual single d.f., returning row number within chunk number.
- Etc.

Software Alchemy: Parallel Computation for the Masses

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.
- Very simple idea:

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.
- Very simple idea:
  - Break the data into disjoint chunks.
  - Apply the estimator to each chunk, getting $\widehat{\theta}_i$ for chunk i.
  - Average the $\widehat{\theta}_i$ to get overall $\widehat{\theta}$.

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.
- Very simple idea:
  - Break the data into disjoint chunks.
  - Apply the estimator to each chunk, getting $\widehat{\theta}_i$ for chunk i.
  - Average the $\widehat{\theta}_i$ to get overall $\widehat{\theta}$.
  - For ML classification algs, "vote" among chunks.

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.
- Very simple idea:
  - Break the data into disjoint chunks.
  - Apply the estimator to each chunk, getting $\widehat{\theta}_i$ for chunk i.
  - Average the $\widehat{\theta}_i$ to get overall $\widehat{\theta}$.
  - For ML classification algs, "vote" among chunks.
- Converts non-EP to stat. equivalent EP.

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.
- Very simple idea:
  - Break the data into disjoint chunks.
  - Apply the estimator to each chunk, getting $\widehat{\theta}_i$ for chunk i.
  - Average the $\widehat{\theta}_i$ to get overall $\widehat{\theta}$.
  - For ML classification algs, "vote" among chunks.
- Converts non-EP to stat. equivalent EP. Thus easy parallelization, possibly even superlinear speedup.

Software Alchemy: Parallel Computation for the Masses

- I call this approach Software Alchemy (SA) (Matloff, JSS, 2016). Method independently proposed by several authors.
- Very simple idea:
  - Break the data into disjoint chunks.
  - Apply the estimator to each chunk, getting $\widehat{\theta}_i$ for chunk i.
  - Average the $\widehat{\theta}_i$ to get overall $\widehat{\theta}$.
  - For ML classification algs, "vote" among chunks.
- Converts non-EP to stat. equivalent EP. Thus easy parallelization, possibly even superlinear speedup.
- The **partools** package has a number of SA ops available.

SA for Time Series

SA for Time Series

- Not i.i.d. but stationarity and finite memory should be enough to prove that it works.

SA for Time Series

- Not i.i.d. but stationarity and finite memory should be enough to prove that it works.
- Should work for ARMA, ARIMA, GARCH, etc.

SA for Time Series

- Not i.i.d. but stationarity and finite memory should be enough to prove that it works.
- Should work for ARMA, ARIMA, GARCH, etc.
- All this should be considered preliminary.

Example

Example

```
library(TSA)
z ← garch.sim(alpha=c(.01,0.9),n=5000000)
system.time(print(garch(z)))
# Coefficient(s):
#          a0          a1          b1
# 1.001e-02   8.980e-01   3.775e-12
#   13.088      0.140    13.228
cls ← makeCluster(2)
setclsinfo(cls)
distribsplit(cls,'z')
system.time(zc2 ← clusterEvalQ(cls, garch(z)$coef))
# 0.000     0.000     5.925
Reduce('+',zc2) / 2
#             a0              a1              b1
# 1.004293e-02  8.910964e-01  3.120723e-05
```

Example

```
library(TSA)
z <- garch.sim(alpha=c(.01,0.9),n=5000000)
system.time(print(garch(z)))
# Coefficient(s):
#         a0          a1          b1
# 1.001e-02   8.980e-01   3.775e-12
#   13.088      0.140    13.228
cls <- makeCluster(2)
setclsinfo(cls)
distribsplit(cls,'z')
system.time(zc2 <- clusterEvalQ(cls, garch(z)$coef))
# 0.000     0.000     5.925
Reduce('+',zc2) / 2
#           a0             a1             b1
# 1.004293e-02  8.910964e-01  3.120723e-05
```

SA version pretty good, 2X speed with coeffs close.

Example

```
library(TSA)
z <- garch.sim(alpha=c(.01,0.9),n=5000000)
system.time(print(garch(z)))
# Coefficient(s):
#          a0          a1          b1
# 1.001e-02   8.980e-01   3.775e-12
#   13.088      0.140    13.228
cls <- makeCluster(2)
setclsinfo(cls)
distribsplit(cls,'z')
system.time(zc2 <- clusterEvalQ(cls, garch(z)$coef))
# 0.000      0.000      5.925
Reduce('+',zc2) / 2
#            a0              a1              b1
# 1.004293e-02  8.910964e-01  3.120723e-05
```

SA version pretty good, 2X speed with coeffs close. But a 4-worker run gave 0.83 for **a1**, a bit further off.

Example

```
library(TSA)
z ← garch.sim(alpha=c(.01,0.9),n=5000000)
system.time(print(garch(z)))
# Coefficient(s):
#          a0          a1          b1
# 1.001e-02   8.980e-01   3.775e-12
#  13.088     0.140    13.228
cls ← makeCluster(2)
setclsinfo(cls)
distribsplit(cls,'z')
system.time(zc2 ← clusterEvalQ(cls, garch(z)$coef))
# 0.000    0.000    5.925
Reduce('+',zc2) / 2
#            a0              a1              b1
# 1.004293e-02  8.910964e-01  3.120723e-05
```

SA version pretty good, 2X speed with coeffs close. But a 4-worker run gave 0.83 for **a1**, a bit further off.
More study needed!

Conclusions

Conclusions

No "silver bullet."

Conclusions

No "silver bullet." But the following should go a long way toward your need for parallel computation.

Conclusions

No “silver bullet.” But the following should go a long way toward your need for parallel computation.

- “Leave it there” and distributed objects/files.

Conclusions

No "silver bullet." But the following should go a long way toward your need for parallel computation.

- "Leave it there" and distributed objects/files.
- SA, previously shown to work well on i.i.d. shows promise time series.

Conclusions

No “silver bullet.” But the following should go a long way toward your need for parallel computation.

- “Leave it there” and distributed objects/files.
- SA, previously shown to work well on i.i.d. shows promise time series.
- The **partools** package adds a lot of convenience.

Conclusions

No "silver bullet." But the following should go a long way toward your need for parallel computation.

- "Leave it there" and distributed objects/files.
- SA, previously shown to work well on i.i.d. shows promise time series.
- The **partools** package adds a lot of convenience.

Ready for the dissent. :-)

Conclusions

No "silver bullet." But the following should go a long way toward your need for parallel computation.

- "Leave it there" and distributed objects/files.
- SA, previously shown to work well on i.i.d. shows promise time series.
- The **partools** package adds a lot of convenience.

Ready for the dissent. :-)

And sorry if I have omitted your favorite software. Just let me know. :-)