Programming with Big Data in R

Drew Schmidt and George Ostrouchov

http://r-pbd.org/

November 19, 2013

The pbdR Core Team

Wei-Chen Chen[1]
George Ostrouchov[1,2]
Pragneshkumar Patel[1]
Drew Schmidt[1]

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 NICS-RDAV center. This work used resources of the Newton HPC Program at the University of Tennessee, Knoxville. 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.

[1]University of Tennessee. Supported in part by the project "NICS Remote Data Analysis and Visualization Center" funded by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center.

[2]Oak Ridge National Laboratory. Supported in part by the project "Visual Data Exploration and Analysis of Ultra-large Climate Data" funded by U.S. DOE Office of Science under Contract No. DE-AC05-00OR22725.

Contents

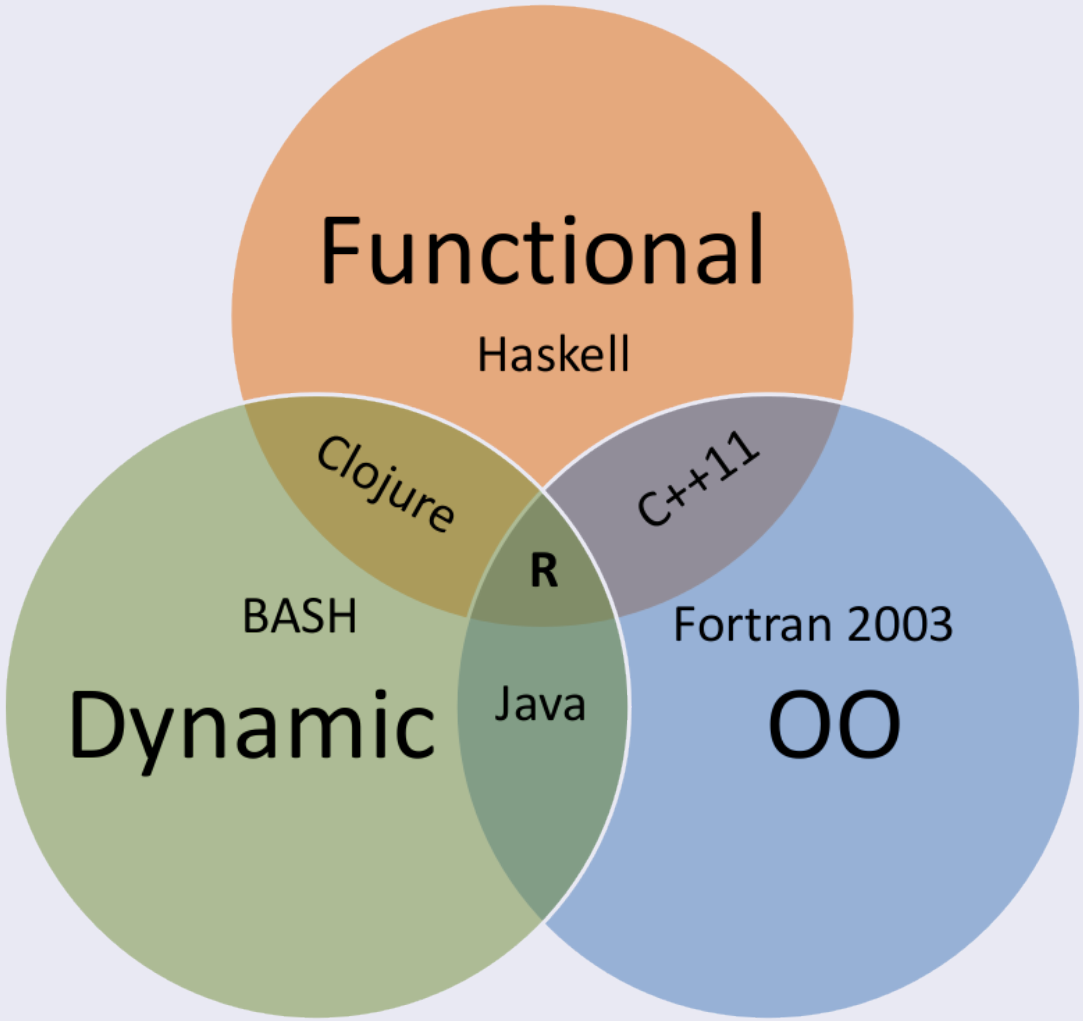
What is R?

- *lingua franca* for data analytics and statistical computing.
- Part programming language, part data analysis package.
- Dialect of S (Bell Labs).
- Syntax designed for data.

Who uses R?


ORACLE
MERCK
KICKSTARTER
Bank of America
okcupid
The New York Times
Shell
Pfizer
ORBITZ
mozilla FOUNDATION
LLOYD'S
Google
ebay
bing
Ford

Language Paradigms



Why R?

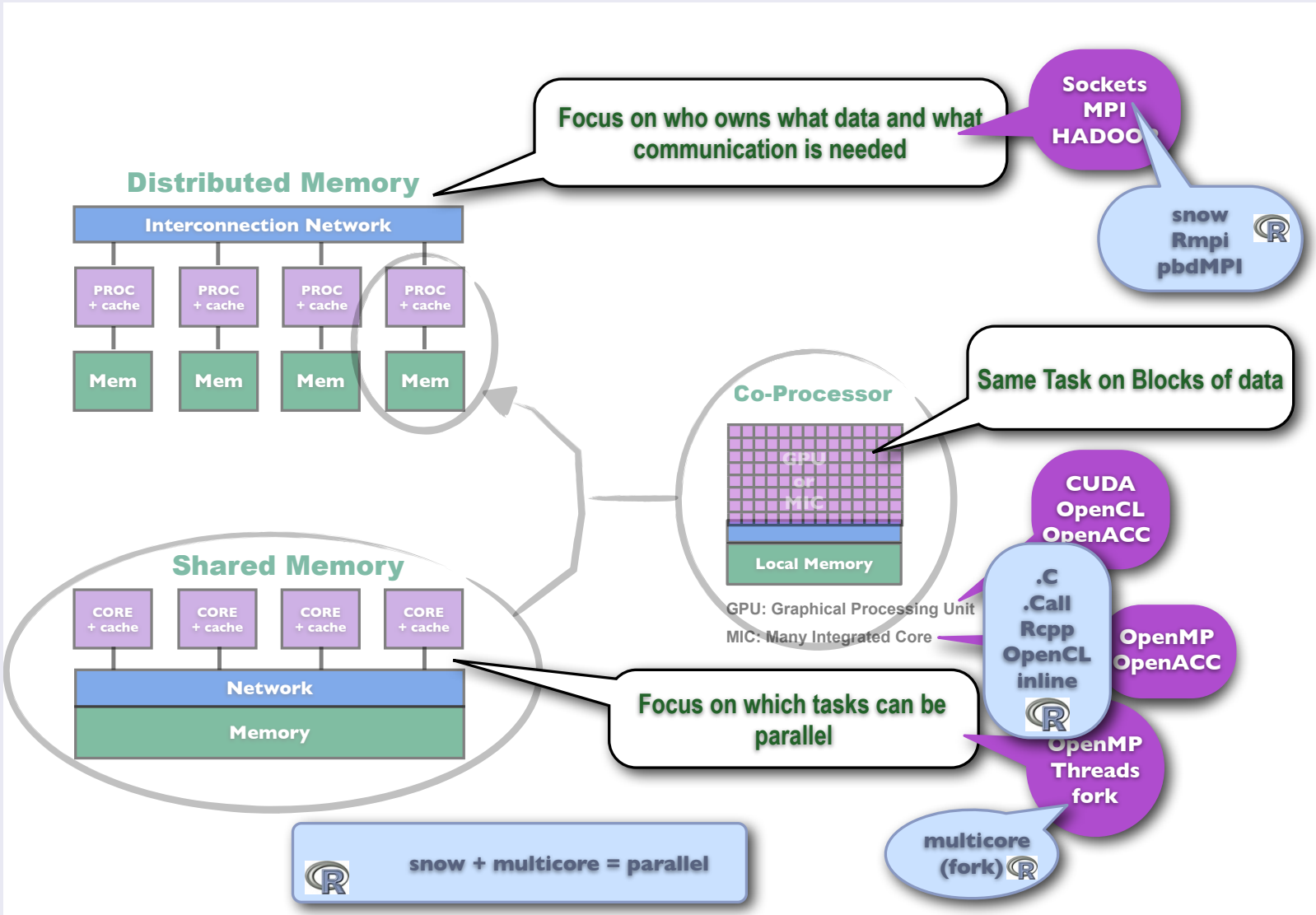
1. People love it.
2. Highly extensible (CRAN $\approx$ 5000 actively maintained packages.
3. Wealth of diverse analytical methods.
4. HPC community has growing need for data analytics.

Problems with R

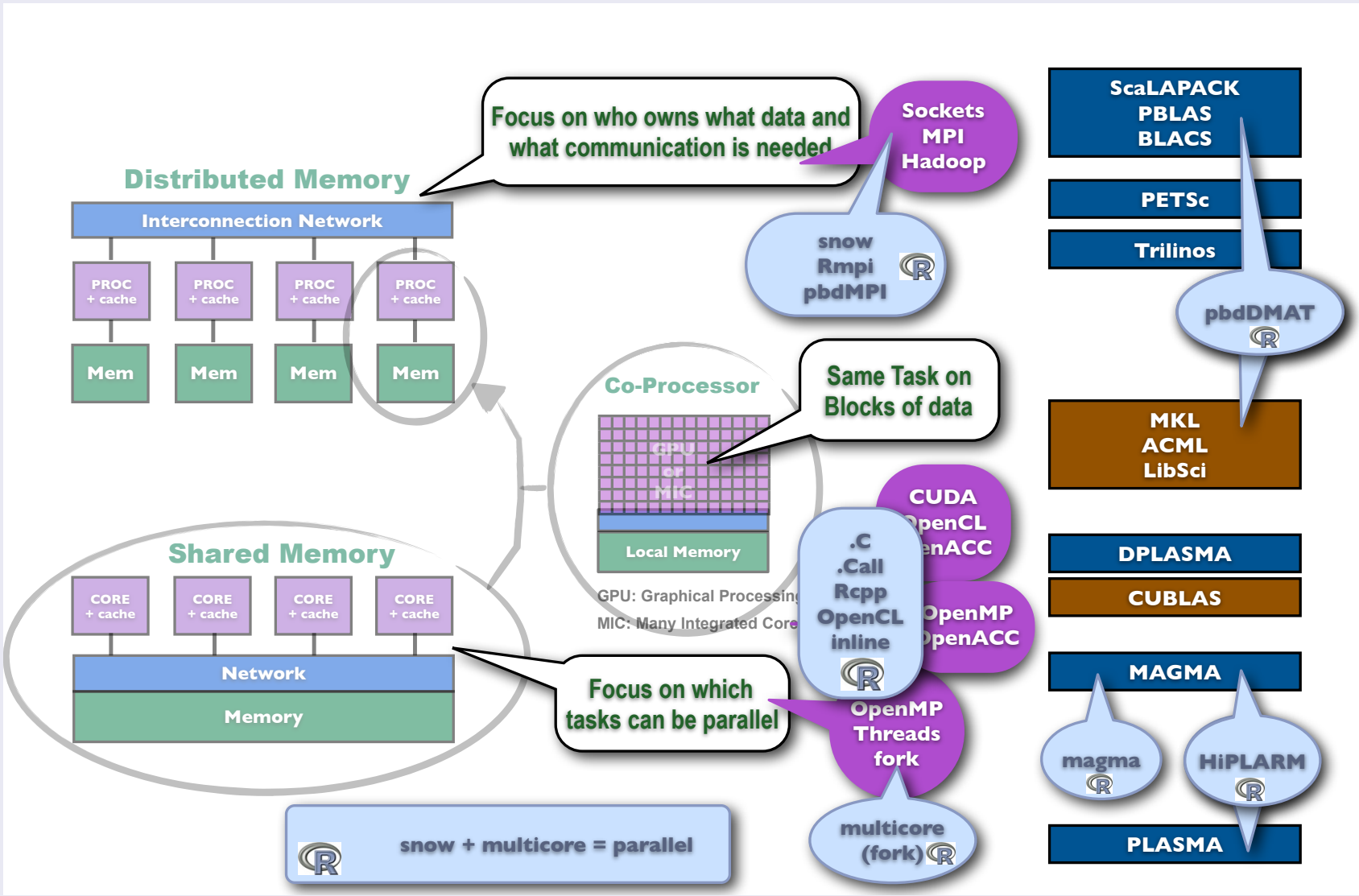
1. Slow.
2. If you don't know what you're doing, it's *really* slow.
3. Data science community has growing data size problem.
4. Chokes on big data.
5. Parallelism: "batteries not included"

In spite of all this. . .

R Interfaces to Native Tools



R Interfaces to Scalable Math Libraries



Contents

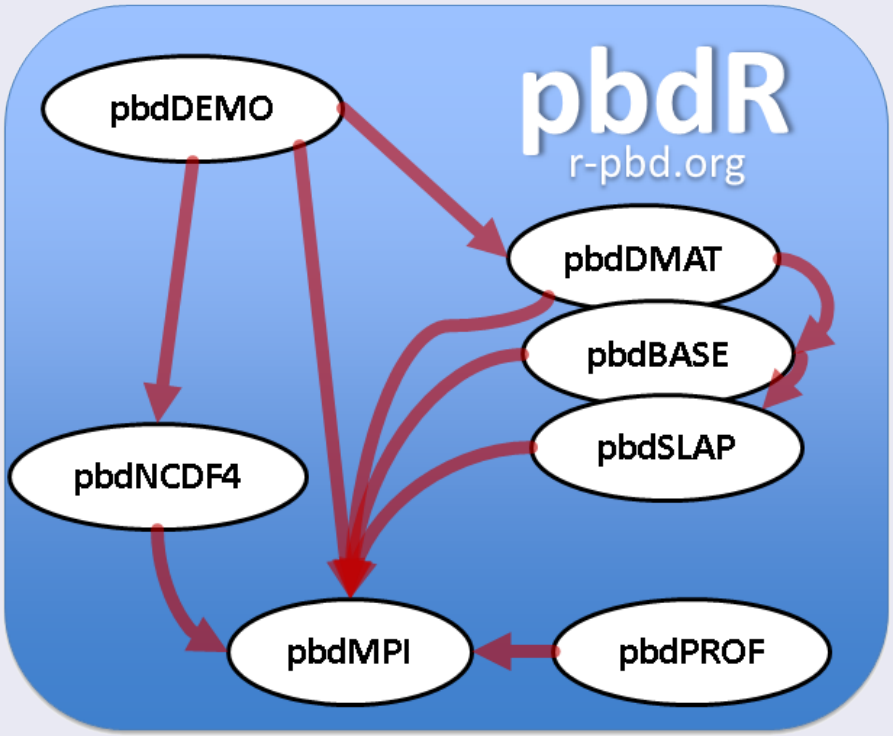
Programming with Big Data in R (pbdR)

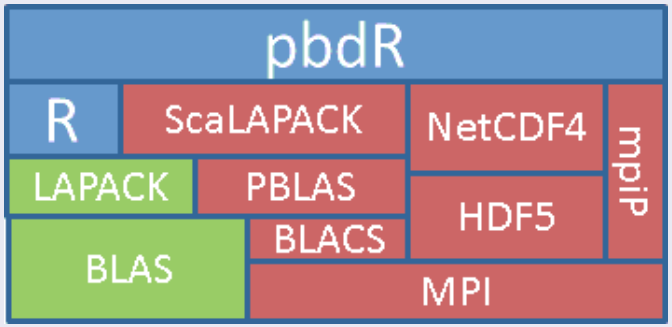
Productivity, Portability, Performance

- *Free*[a] R packages.
- Bridging high-performance C with high-productivity of R
- Distributed data details implicitly managed.
- Methods have syntax *identical* to R.

[a] MPL, BSD, and GPL licensed

pbdR Packages





pbdR Example Syntax

```
1 x <- x[-1, 2:5]
2 x <- log(abs(x) + 1)
3 xtx <- t(x) %*% x
4 ans <- svd(solve(xtx))
```

The above runs on 1 core with R or 10,000 cores with **pbdR**

Profiling with **pbdPROF**

1. Rebuild **pbdR** packages

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

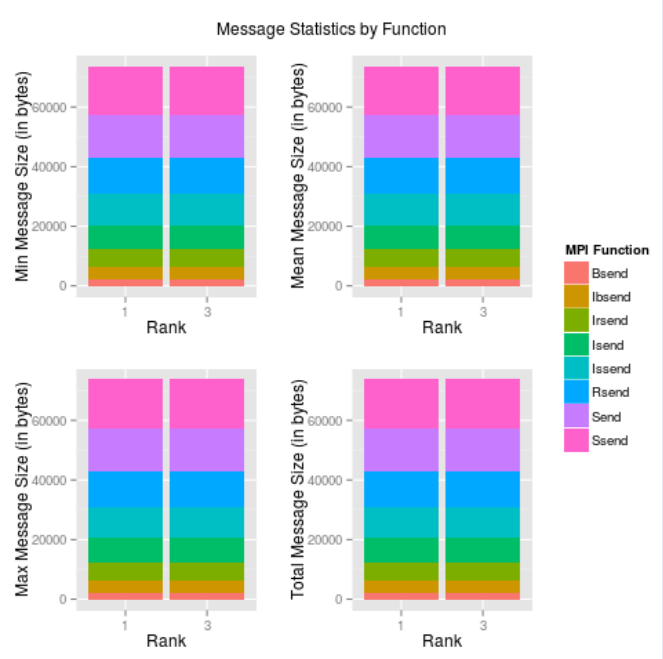
2. Run code

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

3. Analyze results

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

Publication-quality graphs



pbdR on HPC Resources

University of Tennessee

- Kraken (XSEDE)
- Nautilus
- Darter
- Newton

Oak Ridge National Lab

- Titan
- Lens
- Chester
- Sith

Other Resources

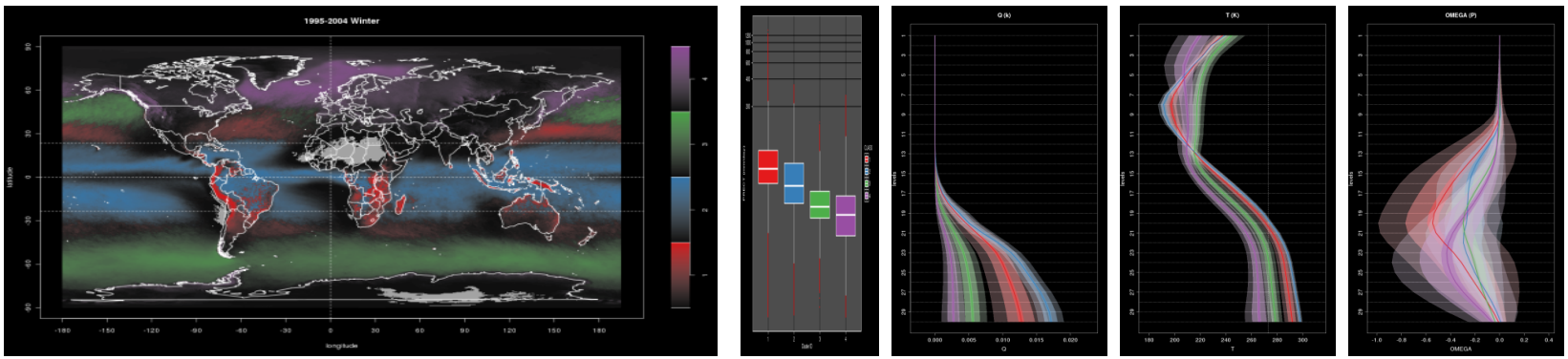
- Stampede, TACC (XSEDE)
- tara, UMBC
- Hopper, NERSC
- Edison, NERSC

If you are interested in installing pbdR: RBigData@gmail.com

Exploration of Climate Data with pbdR

- **2013 INCITE "Attributing Changes in the Risk of Extreme Weather and Climate" Michael Wehner, PI**
  - High resolution atmospheric model, CAM 5.1 (~25 km resolution)
  - Resolves processes responsible for extreme precipitation and storms
- **R and pbdR used for scalable analytics and graphics**
  - Advanced clustering capability (EM algorithm for Gaussian mixture models)

Behavior of extreme precipitation across atmospheric layers of moisture, temperature, and wind



W.-C. Chen, G. Ostrouchov, D. Pugmire, Prabhat, M. Wehner. A Parallel EM Algorithm for Model-Based Clustering Applied to the Exploration of Large Spatio-Temporal Data. *Technometrics* (online, in print: Fall 2013).

OAK RIDGE National Laboratory

Contents

Non-Optimal Choices Throughout

1. Only libre software used (no MKL, ACML, etc.).
2. 1 core = 1 MPI process.
3. No tuning for data distribution.

Benchmark Data

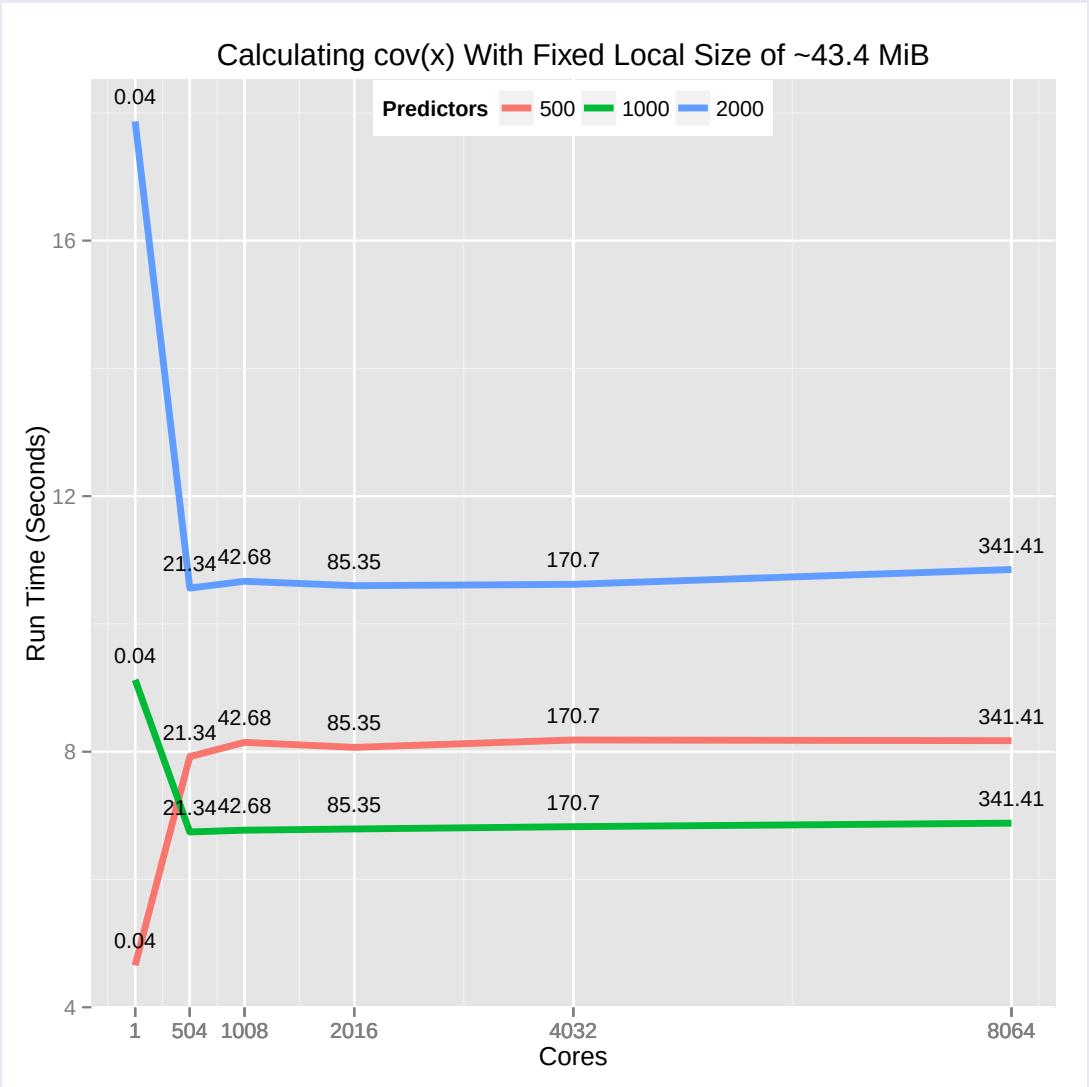
1. Measure wallclock time for covariance and linear regression.
2. Random normal $N(100, 10000)$.
3. Local problem size of $\approx 43.4 MiB$.
4. Three sets: 500, 1000, and 2000 columns.
5. Several runs at different core sizes within each set.

Covariance Code

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

```
1 x <- ddmatrix("rnorm", nrow=n, ncol=p, mean=mean, sd=sd)
2
3 cov.x <- cov(x)
```

cov()


Calculating cov(x) With Fixed Local Size of ~43.4 MiB
Predictors 500 1000 2000
Run Time (Seconds)
16
12
8
4
0.04
21.34
42.68
85.35
170.7
341.41
1
504
1008
2016
4032
8064
Cores

Linear Model Code

Find β such that

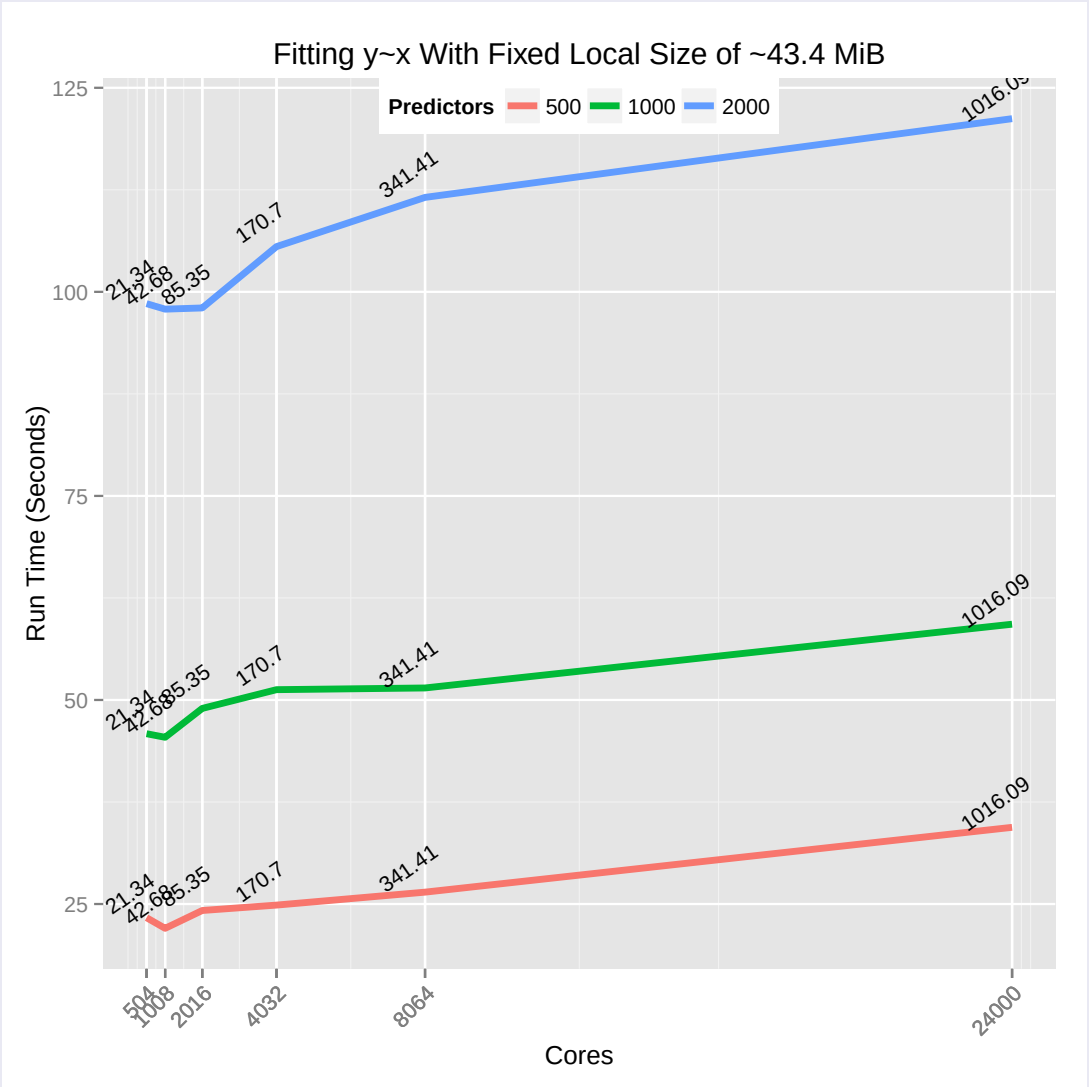
$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

When $\mathbf{X}$ is full rank,

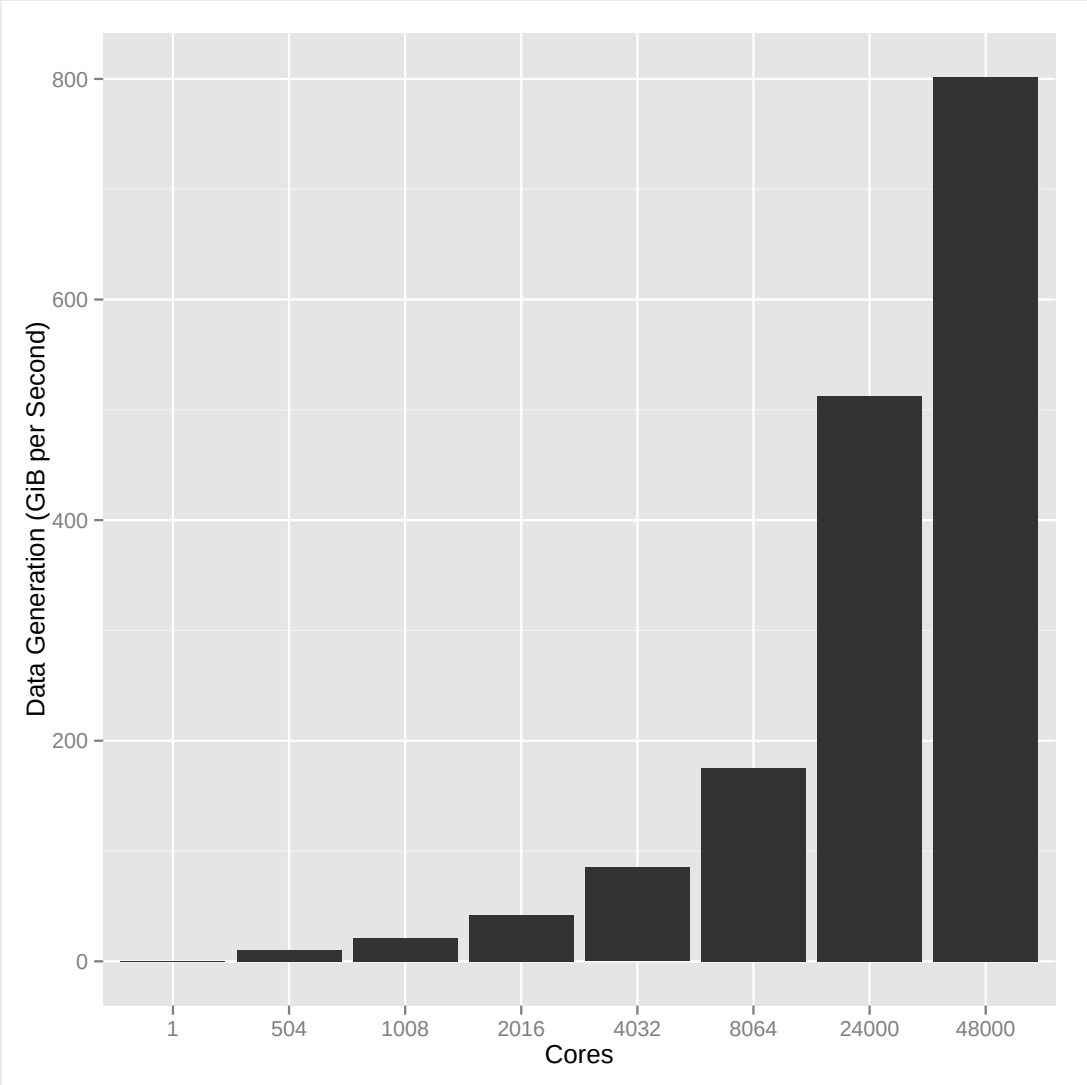
$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

```
1 x <- ddmatrix("rnorm", nrow=n, ncol=p, mean=mean, sd=sd)
2 beta_true <- ddmatrix("runif", nrow=p, ncol=1)
3
4 y <- x %*% beta_true
5
6 beta_est <- lm.fit(x=x, y=y)$coefficients
```

lm.fit()


Fitting y~x With Fixed Local Size of ~43.4 MiB
Predictors 500 1000 2000
Run Time (Seconds)
125
100
75
50
25
21.34
42.68
85.35
170.7
341.41
1016.09
504
1008
2016
4032
8064
24000
Cores

Data Generation



Contents

MPI Operations: The Gang's All Here

- **Communicator wrangling**: `init()`, `finalize()`
- **Rank query**: `comm.rank()`, `comm.size()`
- **Reduction**: `reduce(x, op='sum')`, `allreduce(x)`
- **Gather**: `gather(x)`, `allgather(x)`
- **Broadcast**: `bcast(x)`
- **Barrier**: `barrier()`
- **Send/Receive**: `send(x)`, `recv()`

MPI Operations for R Users

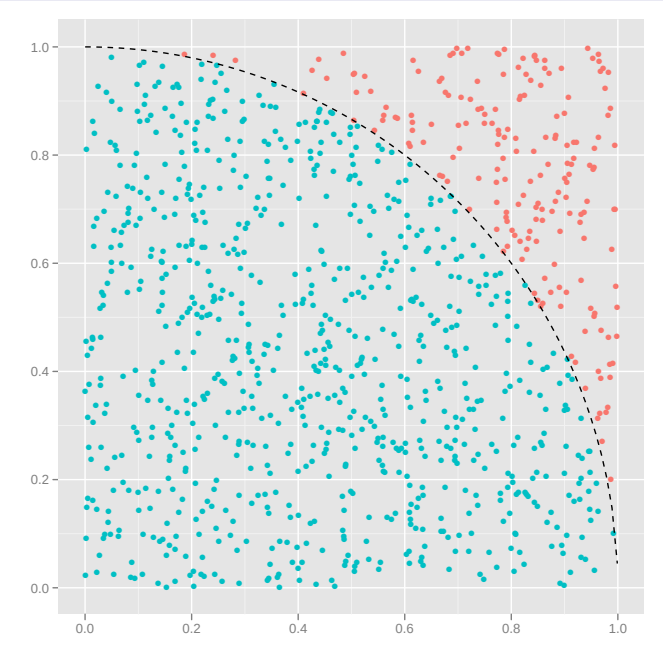
- **Printing**: `comm.print(x)`, `comm.cat(x)`
- **RNG Seeds**: `comm.set.seed(diff=T)`
- **Task Subsetting**: `get.jid(n)`
- ***ply**:

```
pbdApply(X, MARGIN, FUN, ...)
pbdLapply(X, FUN, ...)
pbdSapply(X, FUN, ...)
```

Example 1: Monte Carlo Simulation

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

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



Distributed Matrices

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

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

DMAT: 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 DMAT: Local View

$$\left[\begin{array}{cc|cc} x_{11} & x_{12} & x_{17} & x_{18} \\ x_{21} & x_{22} & x_{27} & x_{28} \\ \hline x_{51} & x_{52} & x_{57} & x_{58} \\ x_{61} & x_{62} & x_{67} & x_{68} \\ \hline x_{91} & x_{92} & x_{97} & x_{98} \end{array}\right]_{5\times4} \left[\begin{array}{cc|c} x_{13} & x_{14} & x_{19} \\ x_{23} & x_{24} & x_{29} \\ \hline x_{53} & x_{54} & x_{59} \\ x_{63} & x_{64} & x_{69} \\ \hline x_{93} & x_{94} & x_{99} \end{array}\right]_{5\times3} \left[\begin{array}{cc} x_{15} & x_{16} \\ x_{25} & x_{26} \\ \hline x_{55} & x_{56} \\ x_{65} & x_{66} \\ \hline x_{95} & x_{96} \end{array}\right]_{5\times2}$$

$$\left[\begin{array}{cc|cc} x_{31} & x_{32} & x_{37} & x_{38} \\ x_{41} & x_{42} & x_{47} & x_{48} \\ \hline x_{71} & x_{72} & x_{77} & x_{78} \\ x_{81} & x_{82} & x_{87} & x_{88} \end{array}\right]_{4\times4} \left[\begin{array}{cc|c} x_{33} & x_{34} & x_{39} \\ x_{43} & x_{44} & x_{49} \\ \hline x_{73} & x_{74} & x_{79} \\ x_{83} & x_{84} & x_{89} \end{array}\right]_{4\times3} \left[\begin{array}{cc} x_{35} & x_{36} \\ x_{45} & x_{46} \\ \hline x_{75} & x_{76} \\ x_{85} & x_{86} \end{array}\right]_{4\times2}$$

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

Randomized SVD[1]

Prototype for Randomized SVD

Given an $m \times n$ matrix $\boldsymbol{A}$, 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 $\boldsymbol{U\Sigma V}^*$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal, and $\boldsymbol{\Sigma}$ is nonnegative and diagonal.

Stage A:

1 Generate an $n \times 2k$ Gaussian test matrix $\boldsymbol{\Omega}$.
2 Form $\boldsymbol{Y} = (\boldsymbol{AA}^*)^q \boldsymbol{A\Omega}$ by multiplying alternately with $\boldsymbol{A}$ and $\boldsymbol{A}^*$.
3 Construct a matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$.

Stage B:

4 Form $\boldsymbol{B} = \boldsymbol{Q}^* \boldsymbol{A}$.
5 Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}} \boldsymbol{\Sigma} \boldsymbol{V}^*$.
6 Set $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.

Note: The computation of $\boldsymbol{Y}$ in step 2 is vulnerable to round-off errors. When high accuracy is required, we must incorporate an orthonormalization step between each application of $\boldsymbol{A}$ and $\boldsymbol{A}^*$; see Algorithm 4.4.

Algorithm 4.4: Randomized Subspace Iteration

Given an $m \times n$ matrix $\boldsymbol{A}$ and integers ℓ and q, this algorithm computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ standard Gaussian matrix $\boldsymbol{\Omega}$.
2 Form $\boldsymbol{Y}_0 = \boldsymbol{A\Omega}$ and compute its QR factorization $\boldsymbol{Y}_0 = \boldsymbol{Q}_0 \boldsymbol{R}_0$.
3 **for** $j = 1, 2, \ldots, q$
4 Form $\widetilde{\boldsymbol{Y}}_j = \boldsymbol{A}^* \boldsymbol{Q}_{j-1}$ and compute its QR factorization $\widetilde{\boldsymbol{Y}}_j = \widetilde{\boldsymbol{Q}}_j \widetilde{\boldsymbol{R}}_j$.
5 Form $\boldsymbol{Y}_j = \boldsymbol{A}\widetilde{\boldsymbol{Q}}_j$ and compute its QR factorization $\boldsymbol{Y}_j = \boldsymbol{Q}_j \boldsymbol{R}_j$.
6 **end**
7 $\boldsymbol{Q} = \boldsymbol{Q}_q$.

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   }
```

[1] Halko 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

Randomized SVD

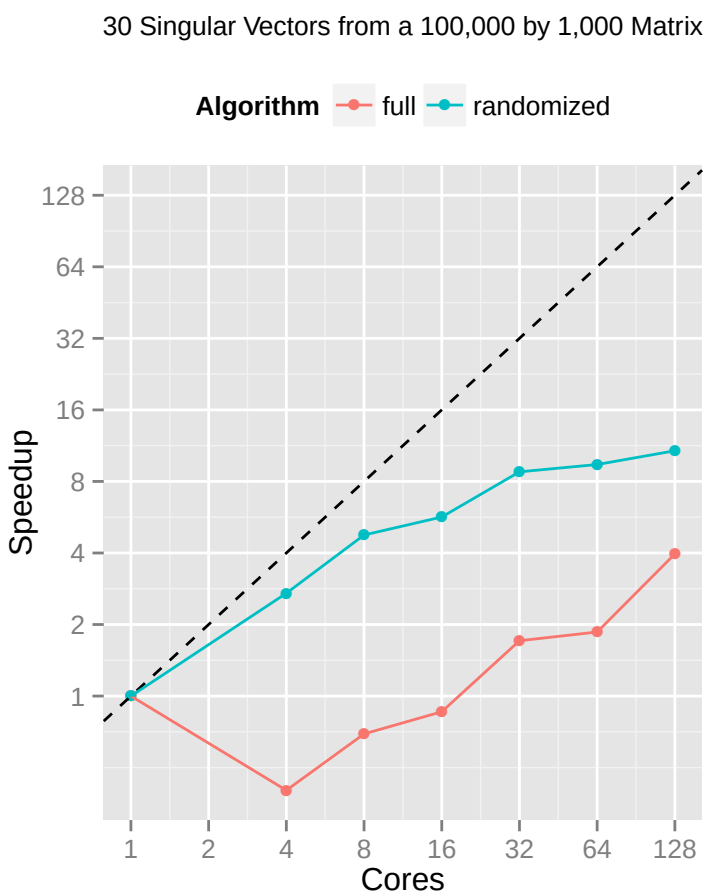
Serial R

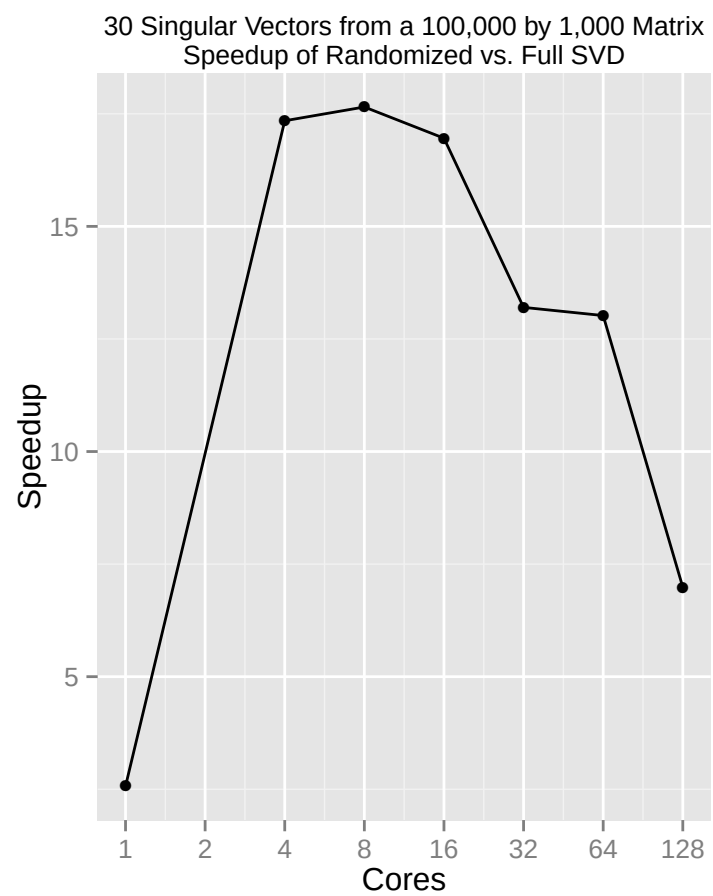
```
1  randSVD <- function(A, k, q=3)
2    {
3      ## Stage A
4      Omega <- matrix(rnorm(n*2*k),
5                      nrow=n, ncol=2*k)
6      Y <- A %*% Omega
7      Q <- qr.Q(qr(Y))
8      At <- t(A)
9      for(i in 1:q)
10       {
11         Y <- At %*% Q
12         Q <- qr.Q(qr(Y))
13         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   }
```

Randomized SVD on ≈ 765 MiB





Thanks for coming!

Questions?

http://r-pbd.org/

Be sure to come to our R BoF: Wednesday 5:30-7:00 room 404