Elevating R to Supercomputers

Drew Schmidt
University of Tennessee, Knoxville

July 12, 2013

Affiliations and Support

The pbdR Core Team
http://r-pbd.org

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

Ostrouchov, Patel, and Schmidt were 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.

Chen and Ostrouchov were 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.

[1] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN

[2] Remote Data Analysis and Visualization Center, University of Tennessee, Knoxville, TN

About This Presentation

Conventions

We use:

- "." as a decimal mark
- "," as order of magnitude separator

Example	Yes	No
One million	1,000,000	1.000.000
One half	0.5	0,5
One thousand and one half	1,000.5	1.000,5

Contents

Why R?

Why R?

1. Because.

Why R?

1. Because.
2. R community has growing data size problem.

Why R?

1. Because.
2. R community has growing data size problem.
3. HPC community has growing need for data analytics.

Elevating R to Supercomputers

Elevating R to Supercomputers

1. Existing code.

Elevating R to Supercomputers

1. Existing code.
2. Syntax.

Elevating R to Supercomputers

1. Existing code.
2. Syntax.
3. Philosophy.

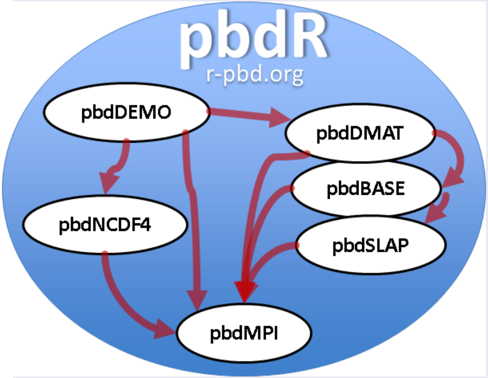
Programming with Big Data in R (pbdR)

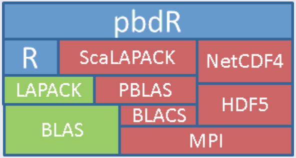
Productivity, Portability, Performance

- *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





pbdMPI vs Rmpi

pbdMPI vs Rmpi

Reduce Operation with Rmpi

```
# int
mpi.allreduce(x, type=1)
# double
mpi.allreduce(x, type=2)
```

Reduce Operation with pbdMPI

```
allreduce(x)
```

pbdMPI vs Rmpi

Reduce Operation with Rmpi

```
# int
mpi.allreduce(x, type=1)
# double
mpi.allreduce(x, type=2)
```

Reduce Operation with pbdMPI

```
allreduce(x)
```

```
> is.integer(1)
[1] FALSE
> is.integer(2)
[1] FALSE
> is.integer(1:2)
[1] TRUE
```

pbdMPI vs Rmpi

Table: Runtimes (seconds) for $10,000 \times 10,000$ `allgather` with **Rmpi** and **pbdMPI**.

Cores	**Rmpi**	**pbdMPI**	Speedup
32	24.6	6.7	3.67
64	25.2	7.1	3.55
128	22.3	7.2	3.10
256	22.4	7.1	3.15

pbdR Example Syntax

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

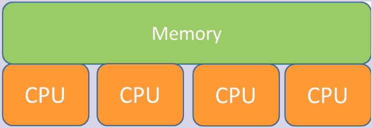
Look familiar?

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

Shared and Distributed Memory Machines

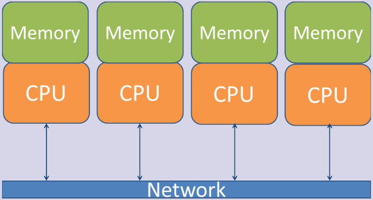
Shared Memory

Direct access to read/change memory (one node)



Distributed

No direct access to read/change memory.



Shared and Distributed Memory Machines

Shared Memory Machines

Thousands of cores

Nautilus, University of Tennessee
1024 cores
4 TB RAM

Distributed Memory Machines

Hundreds of thousands of cores

Kraken, University of Tennessee
112,896 cores
147 TB RAM

Contents

Non-Optimal Choices Throughout

1. Only libre software used (no MKL, ACML, etc.).

Non-Optimal Choices Throughout

1. Only libre software used (no MKL, ACML, etc.).
2. 1 core = 1 MPI process.

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. Random normal $N(100, 10000)$.

Benchmark Data

1. Random normal $N(100, 10000)$.
2. Local problem size of $\approx 43.4 MiB$.

Benchmark Data

1. Random normal $N(100, 10000)$.
2. Local problem size of $\approx 43.4 MiB$.
3. Three sets: 500, 1000, and 2000 columns.

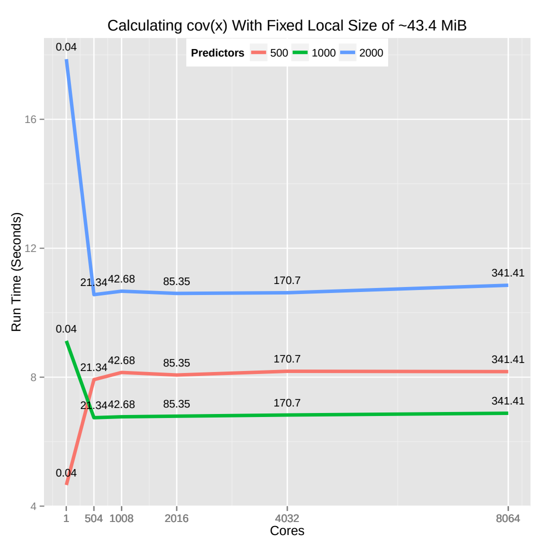
Benchmark Data

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

Covariance Code

```r
x <- ddmatrix("rnorm", nrow=n, ncol=p, mean=mean, sd=sd)

cov.x <- cov(x)
```

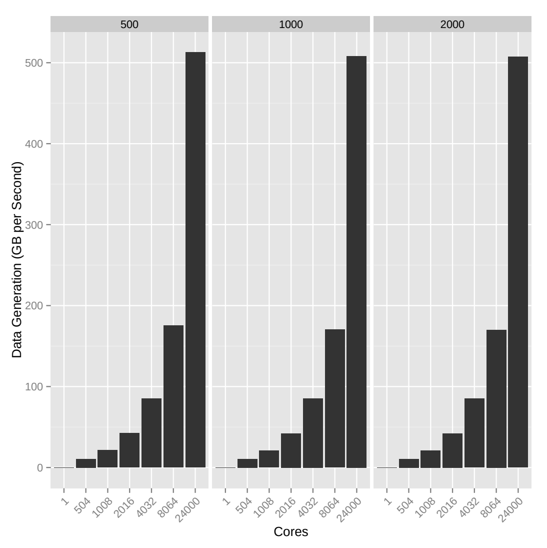
cov()


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
0.04
21.34
42.68
85.35
170.7
341.41
0.04
21.34
42.68
85.35
170.7
341.41
1
504
1008
2016
4032
8064
Cores

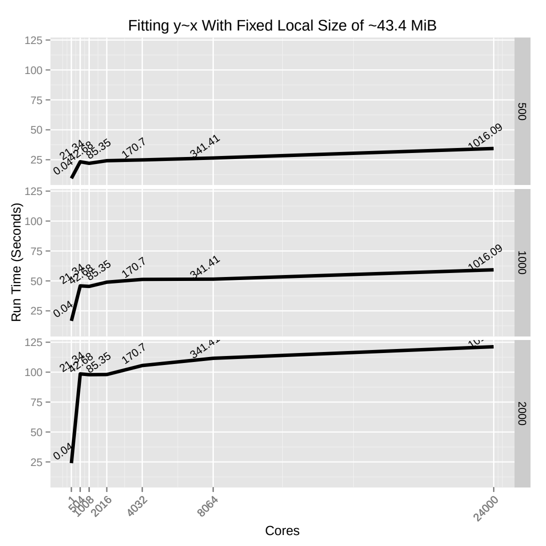
Linear Model Code

```r
x <- ddmatrix("rnorm", nrow=n, ncol=p, mean=mean, sd=sd)
beta_true <- ddmatrix("runif", nrow=p, ncol=1)

y <- x %*% beta_true

beta_est <- lm.fit(x=x, y=y)$coefficients
```

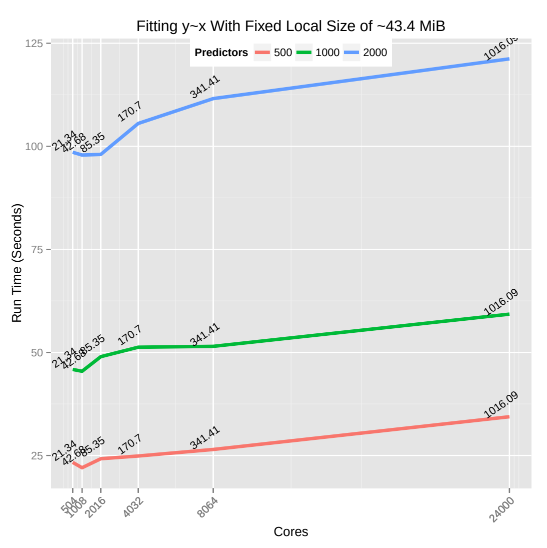
Data Generation



lm.fit()


Fitting y~x With Fixed Local Size of ~43.4 MiB
Run Time (Seconds)
Cores
125
100
75
50
25
500
1000
2000
0.04
21.34
42.68
85.35
170.7
341.41
1016.09
504
1008
2016
4032
8064
24000

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
Cores
504
1008
2016
4032
8064
24000
21.34
42.68
85.35
170.7
341.41
1016.09

Contents

Challenges

- Perceptions.

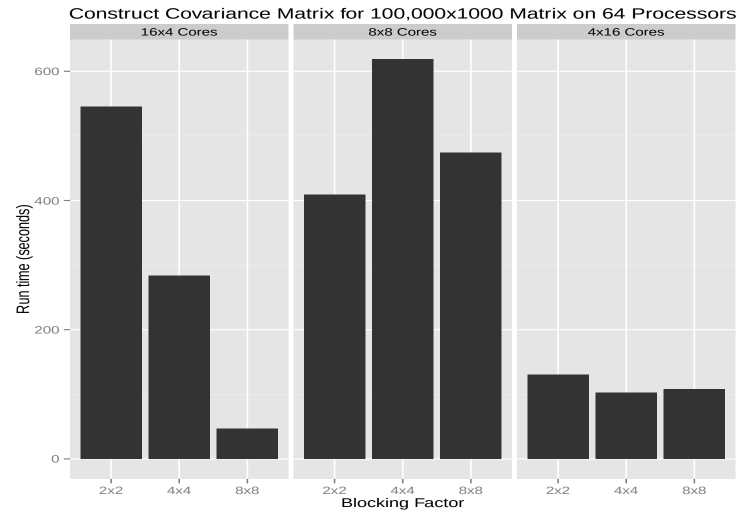
Challenges

- Perceptions.
- Library loading.

Challenges

- Perceptions.
- Library loading.
- Profiling.

Covariance Revisited: Distributed Data Parameter Calibration



Tutorials

- XSEDE13, July 22, San Diego, California, USA
- SC13, November 17-22, Denver, Colorado, USA

Invited Talks

- JSM 2013, August 3-8, Montréal, Québec
- IASC, Aug 22-23, Seoul
- World Statistics Congress, August 25-30, Hong Kong

Thanks for coming!

Questions?

https://github.com/wrathematics/talks/blob/master/user2013/elevatingr.pdf?raw=true