Tall and Skinny QR Factorizations in MapReduce

SAND2011-4037C

David F. Gleich

Paul G. Constantine

MapReduce 2011 Workshop

8 June 2011

Thanks to Mark Hoemmen, Tammy Kolda, Todd Plantenga, Craig Ulmer, Justin Basilico @ Sandia and James Demmel, Gray Ballard, Austin Benson @ Berkeley for helpful discussions.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

MapReduce is great for TSQR!

Data A tall and skinny (TS) matrix by rows

Map QR factorization of local rows

Reduce QR factorization of local rows

Demmel et al. showed that this construction works to compute a QR factorization with *minimal communication*

Input **500,000,000-by-100 matrix**

Each record **1-by-100 row**

HDFS Size **423.3 GB**

Time to compute $\|\boldsymbol{A}\mathbf{e}_i\|$ (the norm of each column) **161 sec.**

Time to compute $\boldsymbol{R}$ in qr($\boldsymbol{A}$) **387 sec.**

```
git clone https://github.com/dgleich/mrtsqr
```

On a 64-node Hadoop cluster with 4x2TB, one Core i7-920, 12GB RAM/node

Outline

Tall and Skinny QR factorizations

Implementation in MapReduce

Synthetic experiments for performance

Real-world test: Tinyimages

Simulation Informatics

QR Factorization

Let $\boldsymbol{A} : m \times n, m \geq n$, real

$$\boldsymbol{A} = \boldsymbol{QR}$$

$\boldsymbol{Q}$ is $m \times n$ orthogonal ($\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$)

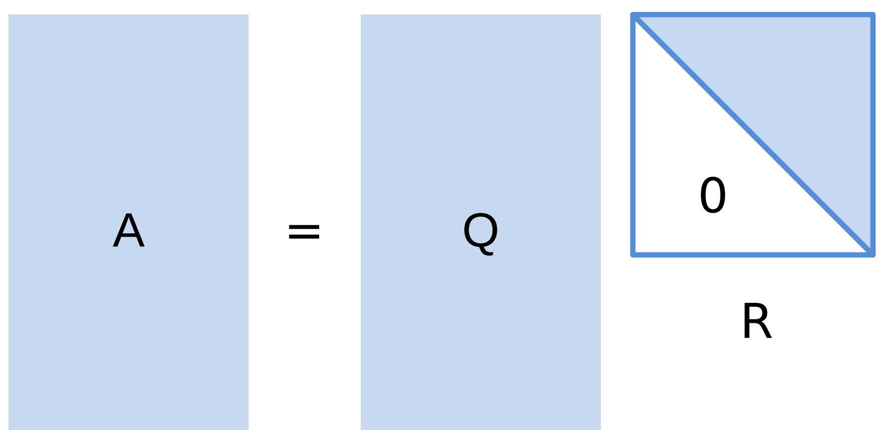
$\boldsymbol{R}$ is $n \times n$ upper triangular.

Using QR for regression

$\min \|\mathbf{b} - \boldsymbol{A}\mathbf{x}\|$ is given by the solution of $\boldsymbol{R}\mathbf{x} = \boldsymbol{Q}^T\mathbf{b}$

QR is block normalization

"normalize" a vector usually generalizes to computing $\boldsymbol{Q}$ in the QR



Tall and Skinny QR factorizations

A

Tall and Skinny matrices
$m \gg n$ arise in

regression with many samples

block iterative methods

panel factorizations

model reduction problems

general linear models
with many samples

tall-and-skinny SVD/PCA

all of these applications
need a QR factorization
some only need $\boldsymbol{R}$!

Normal Equations

Use $\boldsymbol{A}^T\boldsymbol{A}$? Yields only $O(\sqrt{\varepsilon})$ accuracy.

Communication avoiding TSQR

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \boldsymbol{A}_3 \\ \boldsymbol{A}_4 \end{bmatrix} \qquad \boldsymbol{A} = \underbrace{\begin{bmatrix} \boldsymbol{Q}_1 & & & \\ & \boldsymbol{Q}_2 & & \\ & & \boldsymbol{Q}_3 & \\ & & & \boldsymbol{Q}_4 \end{bmatrix}}_{8n \times 4n} \underbrace{\begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{R}_2 \\ \boldsymbol{R}_3 \\ \boldsymbol{R}_4 \end{bmatrix}}_{4n \times n}$$

First, do QR factorizations of each local matrix $\boldsymbol{A}_i$

Second, compute a QR factorization of the new "R"

$$\boldsymbol{A} = \overbrace{\underbrace{\begin{bmatrix} \boldsymbol{Q}_1 & & & \\ & \boldsymbol{Q}_2 & & \\ & & \boldsymbol{Q}_3 & \\ & & & \boldsymbol{Q}_4 \end{bmatrix}}_{8n \times 4n} \underbrace{\tilde{\boldsymbol{Q}}}_{4n \times n}}^{=\boldsymbol{Q}} \underbrace{\tilde{\boldsymbol{R}}}_{n \times n}$$

Demmel et al. 2008. Communicating avoiding parallel and sequential QR.

Communication avoiding TSQR

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix}$$

$$A = \underbrace{\begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & Q_3 & \\ & & & Q_4 \end{bmatrix}}_{8n \times 4n} \underbrace{\begin{bmatrix} R_1 \\ R_2 \\ \hline R_3 \\ R_4 \end{bmatrix}}_{4n \times n} = \underbrace{\begin{bmatrix} Q_5 & \\ \hline & Q_6 \end{bmatrix}}_{4n \times 2n} \underbrace{\begin{bmatrix} R_5 \\ \hline R_6 \end{bmatrix}}_{2n \times n}$$

But we actually do the second step recursively!

$$A = \overbrace{\underbrace{\begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & Q_3 & \\ & & & Q_4 \end{bmatrix}}_{8n \times 4n} \underbrace{\begin{bmatrix} Q_5 & \\ \hline & Q_6 \end{bmatrix}}_{4n \times 2n} \underbrace{Q_7}_{2n \times n}}^{=Q} \underbrace{R_7}_{n \times n}$$

Demmel et al. 2008. Communicating avoiding parallel and sequential QR.

Fully serial TSQR

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \\ \mathbf{A}_4 \end{bmatrix}$$

Compute QR of $\mathbf{A}_1$,
read $\mathbf{A}_2$, update QR, …

$$\mathbf{A}_1 = \mathbf{Q}_1 \mathbf{R}_1; \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{A}_2 \end{bmatrix} = \mathbf{Q}_2 \mathbf{R}_2; \begin{bmatrix} \mathbf{R}_2 \\ \mathbf{A}_3 \end{bmatrix} = \mathbf{Q}_3 \mathbf{R}_3; \begin{bmatrix} \mathbf{R}_3 \\ \mathbf{A}_4 \end{bmatrix} = \mathbf{Q}_4 \mathbf{R}_4$$

$$\mathbf{A} = \overbrace{\underbrace{\begin{bmatrix} \mathbf{Q}_1 & & & \\ & \mathbf{I}_{2n} & & \\ & & \mathbf{I}_{2n} & \\ & & & \mathbf{I}_{2n} \end{bmatrix}}_{8n \times 7n} \underbrace{\begin{bmatrix} \mathbf{Q}_2 & & \\ & \mathbf{I}_{2n} & \\ & & \mathbf{I}_{2n} \end{bmatrix}}_{7n \times 5n} \underbrace{\begin{bmatrix} \mathbf{Q}_3 & \\ & \mathbf{I}_{2n} \end{bmatrix}}_{5n \times 3n} \underbrace{\mathbf{Q}_4}_{3n \times n}}^{= \mathbf{Q}} \underbrace{\mathbf{R}}_{n \times n}$$

Demmel et al. 2008. Communicating avoiding parallel and sequential QR.

More generally, any sequence of QR factorizations in a tree that yield a single R is a valid QR factorization.

In our MapReduce implementation, we combine these two steps!

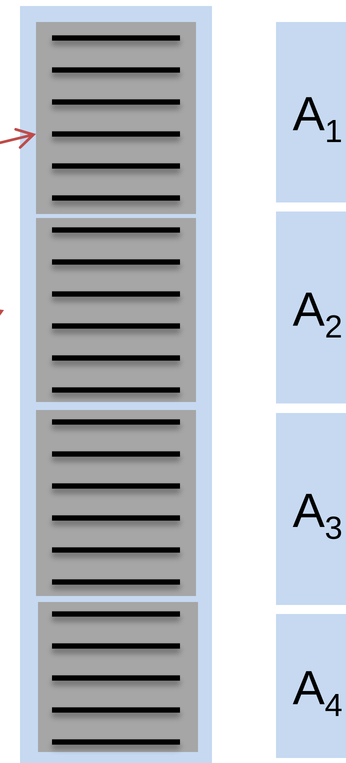
MapReduce matrix storage

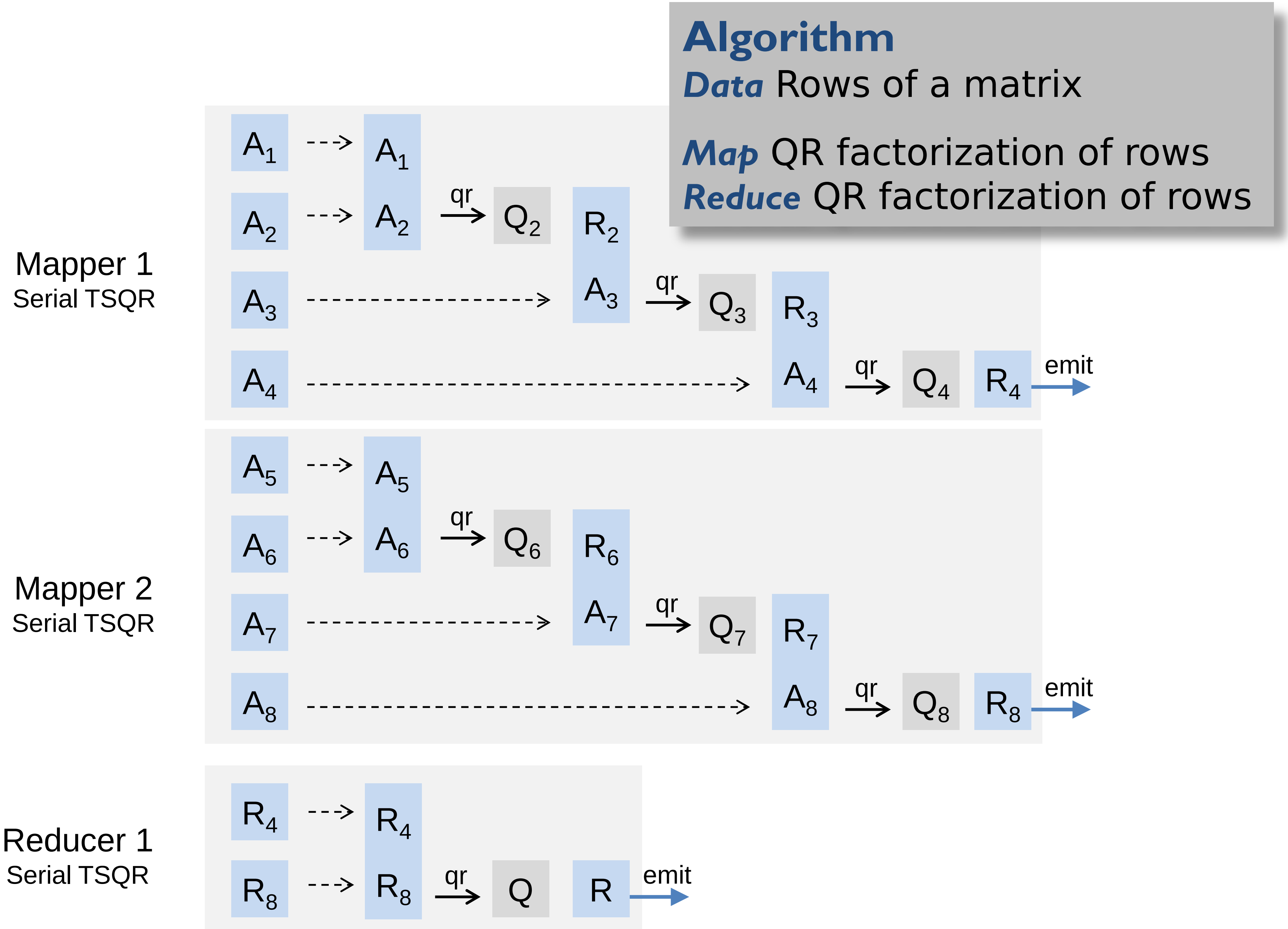
$\boldsymbol{A} : m \times n, m \gg n$

Key is an arbitrary row-id

Value is the $1 \times n$ array for a row.

Each submatrix $\boldsymbol{A}_i$ is an input split.




Algorithm
Data Rows of a matrix
Map QR factorization of rows
Reduce QR factorization of rows
Mapper 1
Serial TSQR
A1
A2
A3
A4
A1
A2
qr
Q2
R2
A3
qr
Q3
R3
A4
qr
Q4
R4
emit
Mapper 2
Serial TSQR
A5
A6
A7
A8
A5
A6
qr
Q6
R6
A7
qr
Q7
R7
A8
qr
Q8
R8
emit
Reducer 1
Serial TSQR
R4
R8
R4
R8
qr
Q
R
emit

Key Limitation

Computes only $\boldsymbol{R}$ and not $\boldsymbol{Q}$

But $\boldsymbol{Q} = \boldsymbol{A}\boldsymbol{R}^{+}$ (pseudo-inverse) with another MapReduce iteration.

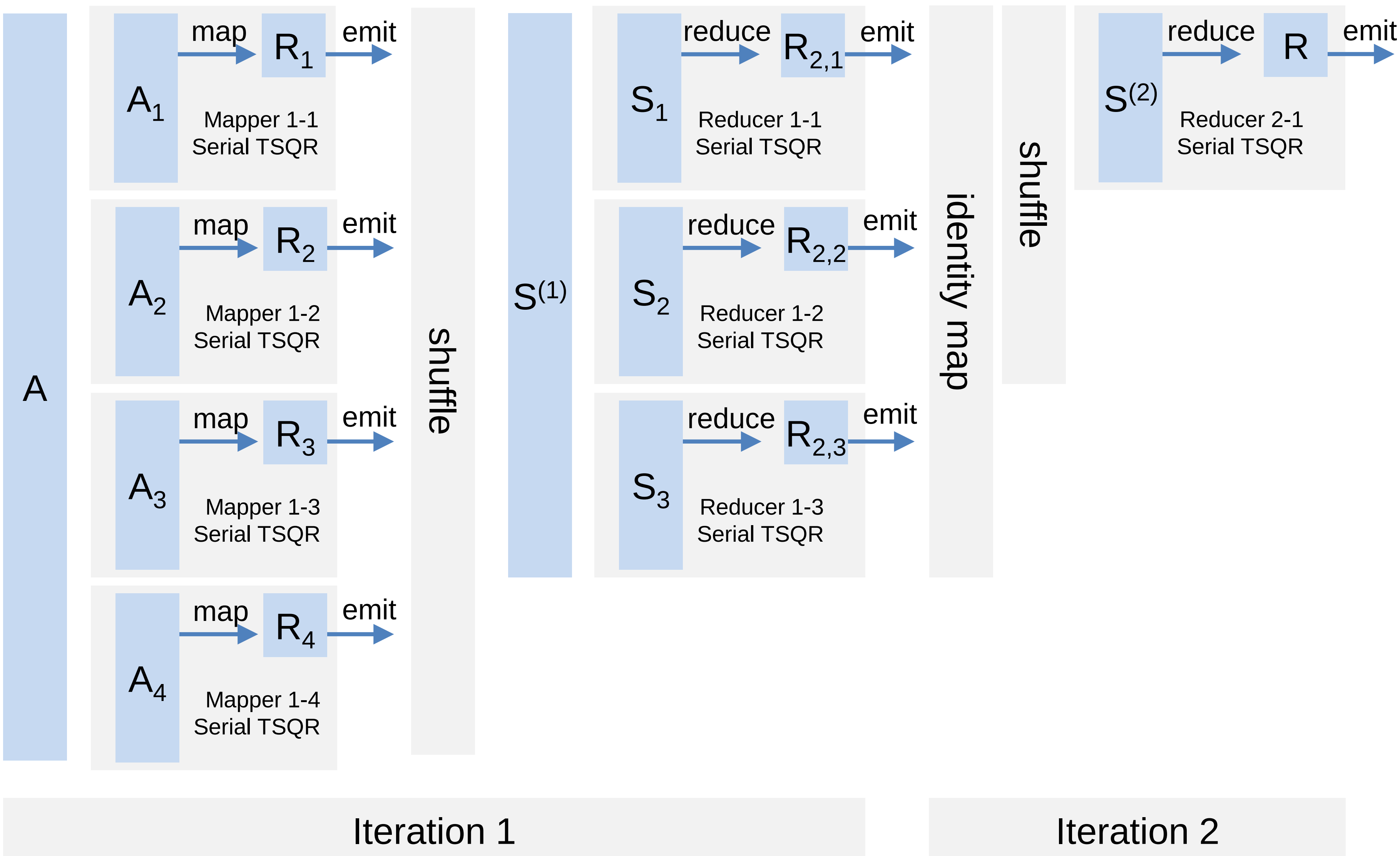
This approach has poor numerical stability, but we (in collaboration with Jim Demmel and Austin Benson) are working on it. (Multiple iterations do better)

We use random keys for new output – makes it hard to store data to recreate Q in a different fashion.

In hadoopy

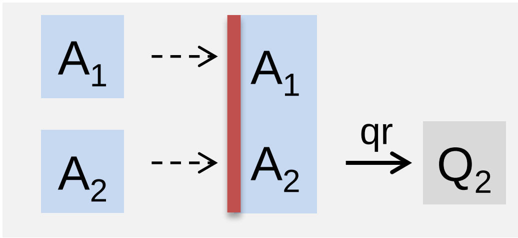
```
import random, numpy, hadoopy
class SerialTSQR:
  def __init__(self,blocksize,isreducer):
    self.bsize=blocksize
    self.data = []
    if isreducer: self.__call__ = self.reducer
    else: self.__call__ = self.mapper

  def compress(self):
    R = numpy.linalg.qr(
          numpy.array(self.data),'r')
    # reset data and re-initialize to R
    self.data = []
    for row in R:
      self.data.append([float(v) for v in row])

  def collect(self,key,value):
    self.data.append(value)
    if len(self.data)>self.bsize*len(self.data[0]):
      self.compress()

  def close(self):
    self.compress()
    for row in self.data:
      key = random.randint(0,2000000000)
      yield key, row

  def mapper(self,key,value):
    self.collect(key,value)

  def reducer(self,key,values):
    for value in values: self.mapper(key,value)

if __name__=='__main__':
  mapper = SerialTSQR(blocksize=3,isreducer=False)
  reducer = SerialTSQR(blocksize=3,isreducer=True)
  hadoopy.run(mapper, reducer)
```

Too many maps? Add an iteration!

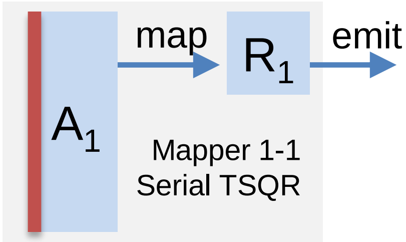


mrtsqr – summary of parameters

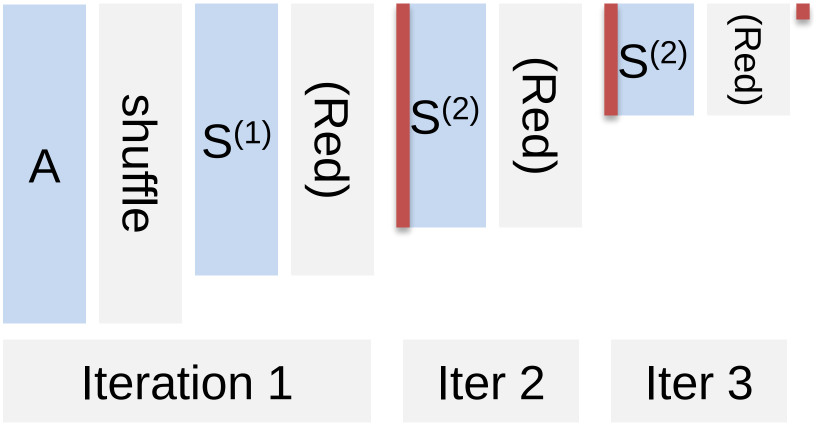
Blocksize How many rows to read before computing a QR factorization, expressed as a multiple of the number of columns (See paper)



Splitsize The size of each local matrix



Reduction tree The number of reducers and iterations to use



Hadoop streaming frameworks

Synthetic data test 100,000,000-by-500 matrix (~500GB)

Codes implemented in MapReduce streaming

Matrix stored as TypedBytes lists of doubles

Python frameworks use Numpy+Atlas

Custom C++ TypedBytes reader/writer with Atlas

New non-streaming Java implementation too

	Iter 1 QR (secs.)	Iter 1 Total (secs.)	Iter 2 Total (secs.)	Overall Total (secs.)
Dumbo	67725	960	217	1177
Hadoopy	70909	612	118	730
C++	**15809**	**350**	**37**	**387**
Java		436	66	502

C++ in streaming beats a native Java implementation.

All timing results from the Hadoop job tracker

Varying splitsize *Synthetic Data*

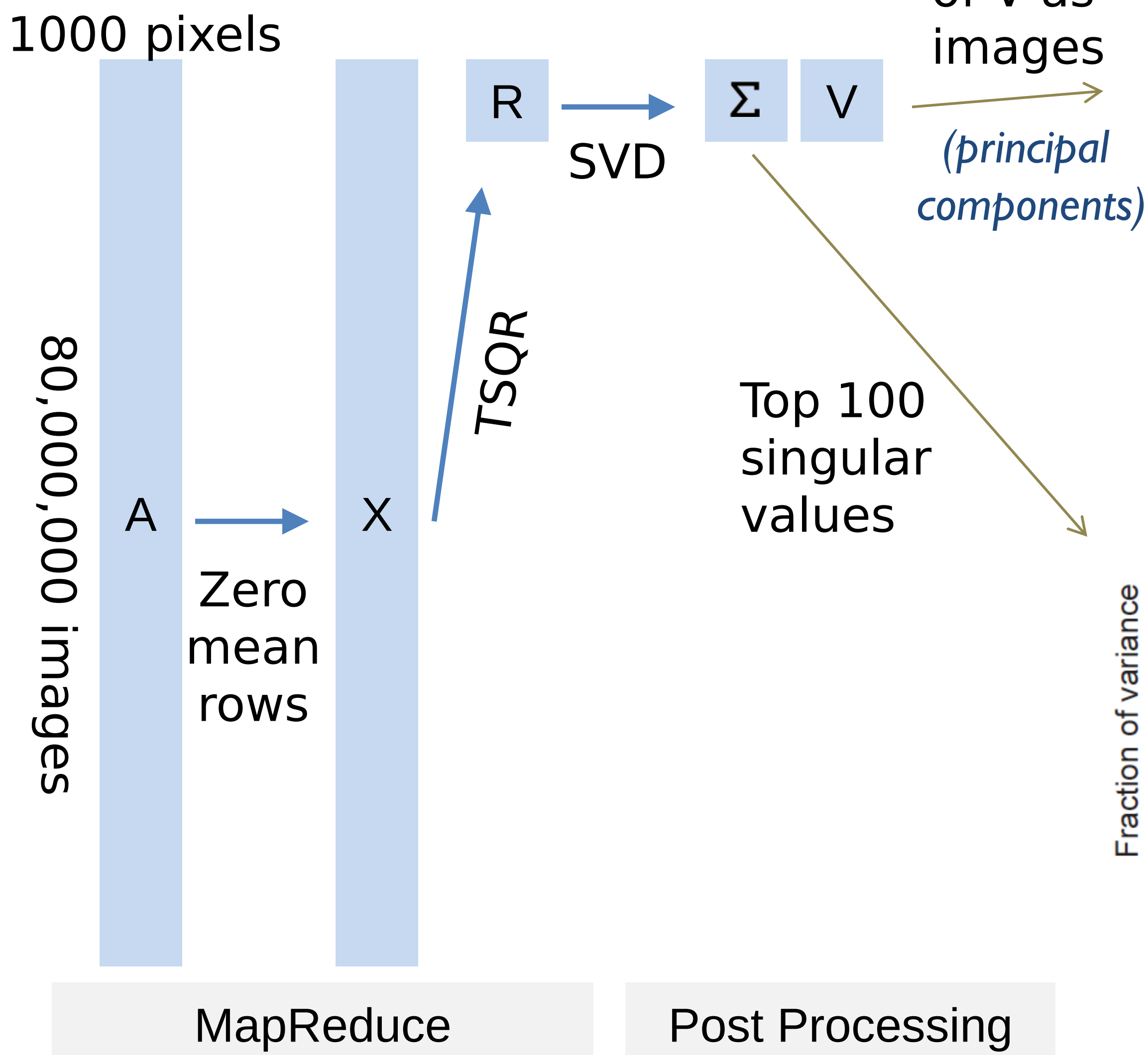
Cols.	*Iters.*	*Split (MB)*	*Maps*	*Secs.*
50	1	64	8000	388
–	–	256	2000	184
–	–	512	1000	**149**
–	2	64	8000	425
–	–	256	2000	220
–	–	512	1000	191
1000	1	512	1000	666
–	2	64	6000	590
–	–	256	2000	432
–	–	512	1000	**337**

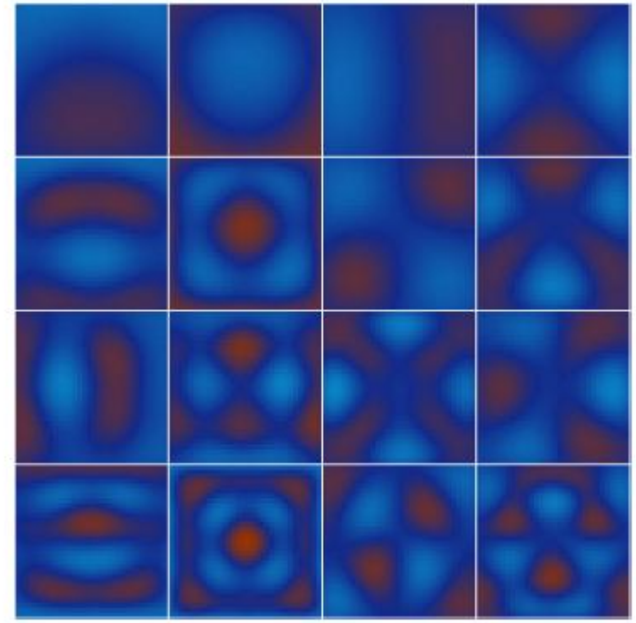
Increasing split size improves performance (accounts for Hadoop data movement)

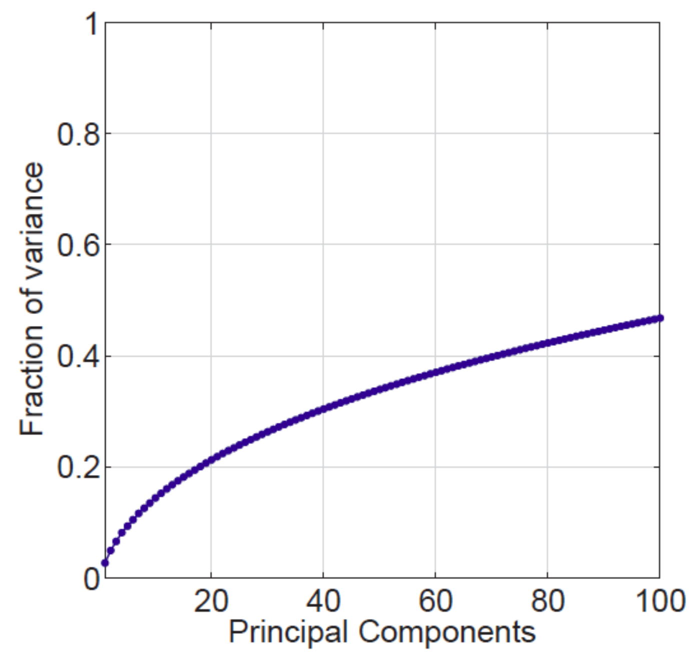
Increasing iterations helps for problems with many columns.

(1000 columns with 64-MB split size overloaded the single reducer.)

Tinyimages PCA







Tiny imges is a 300GB database of images from 10,000 google queries. TSQR took about 30 minutes using dumbo framework

The problem *Simulation ain't cheap!*

21st Century Science in a nutshell

Experiments are not practical / feasible.

Simulate things instead.

But do we trust the simulations?

We're trying!

21.1st Century Science *in a nutshell?*

Simulations are expensive.

Let data provide a surrogate.

Input Parameters

s

Time history of simulation

f

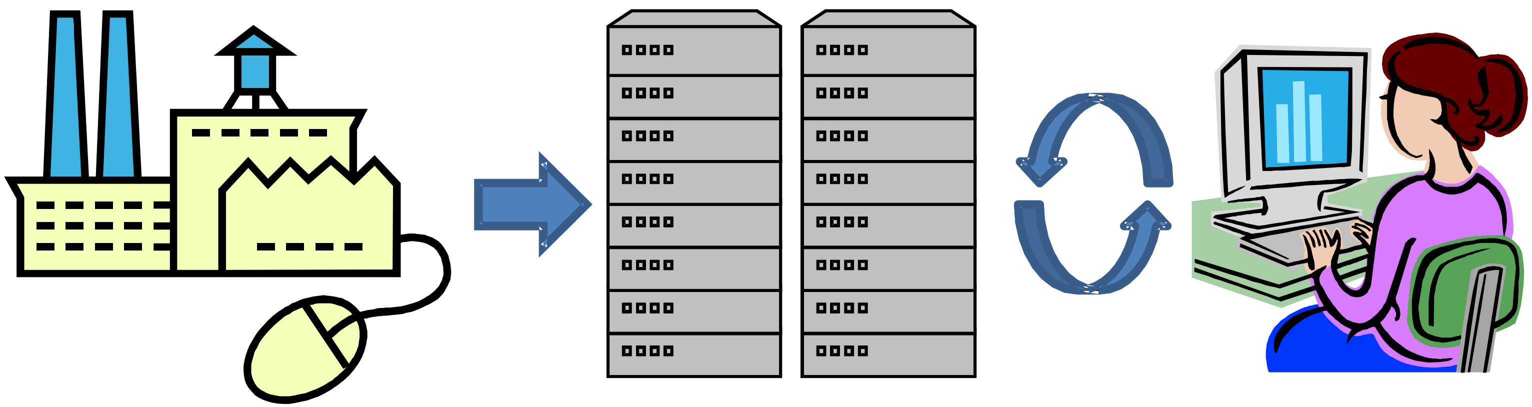
The Database

The idea and vision: Store the runs!

Supercomputer

MapReduce cluster

Engineer



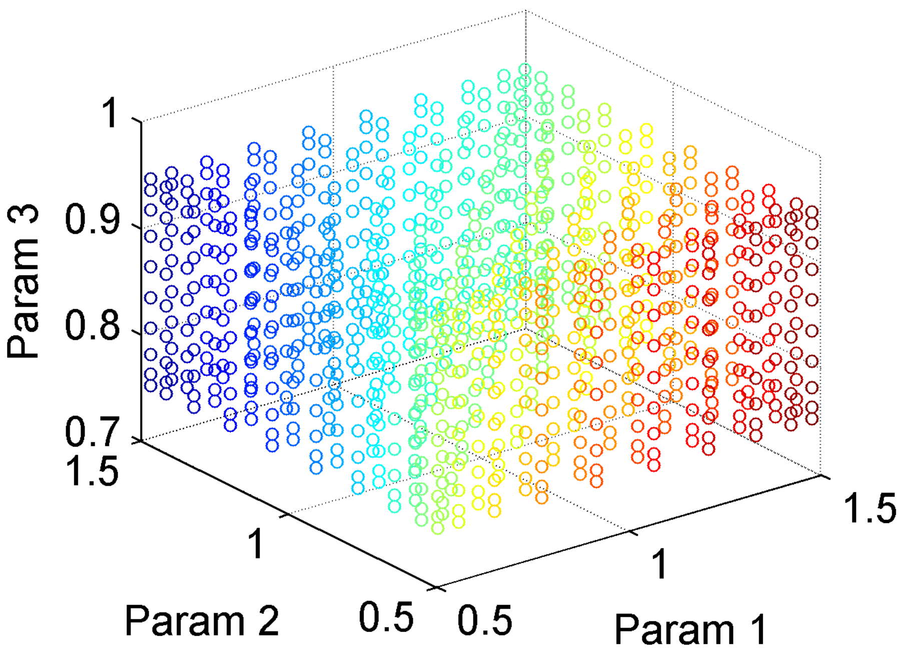
Each multi-day HPC simulation generates gigabytes of data.

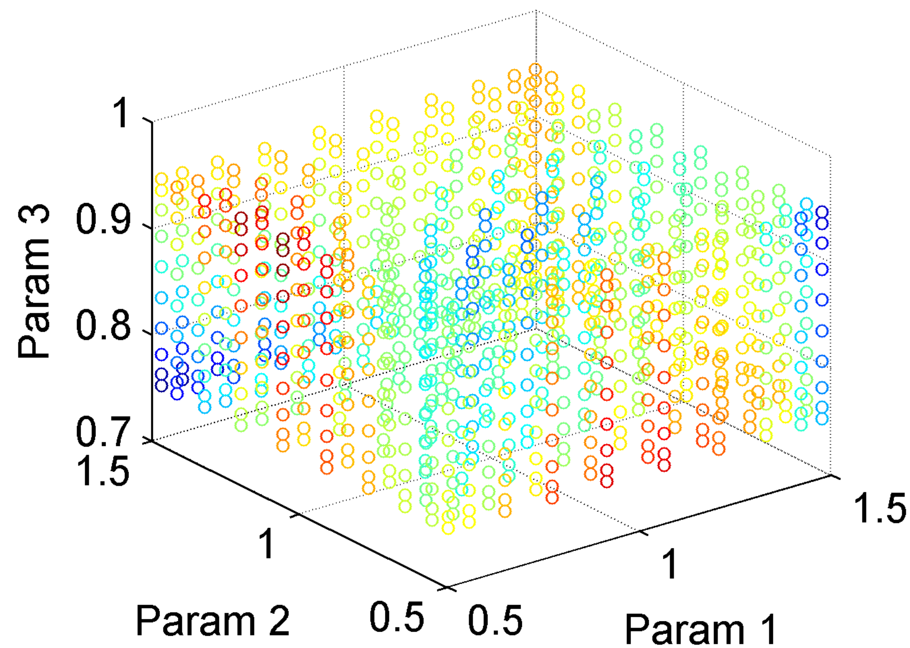
A MapReduce cluster can hold hundreds or thousands of old simulations …

… enabling engineers to query and analyze months of simulation data for statistical studies and uncertainty quantification.

Data driven Green's functions

500GB of simulation data → TSSVD on 6,400,000 x 1000 matrix





The simulation varies most in the corners of the parameter space.

These analyses will help understand which parameters matter, and where to continue running the simulations.

Future work & Desiderata

Round off analysis of MR job for $\boldsymbol{Q}$

Compare against MPI based TSQR *What's a fair way?*

Compare against random sampling

Non-random keys, better partitioners?

A *mathematical reduce*

Serialization for numeric arrays *Roll my own?*

Google mrtsqr gleich

```
git clone https://github.com/dgleich/mrtsqr
```

*Why not **the normal equations?***

Given $\boldsymbol{A}$

compute $\boldsymbol{B} = \boldsymbol{A}^T \boldsymbol{A}$

then $\boldsymbol{B} = \boldsymbol{R}^T \boldsymbol{R}$ with a Cholesky factorization

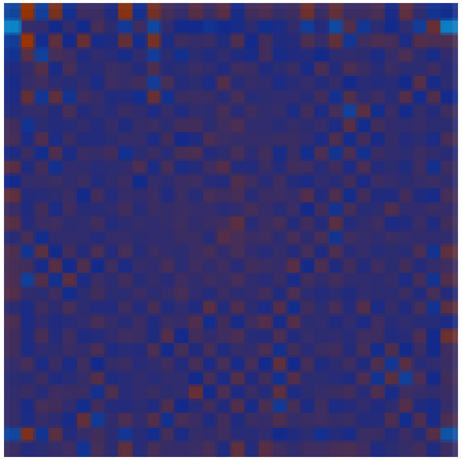
And it's the same $\boldsymbol{R}$ as in the QR!

But, you only get $O(\sqrt{\varepsilon})$ accuracy vs. $O(\varepsilon)$ accuracy for QR.

(ε is the machine precision)

Tinyimages Regression

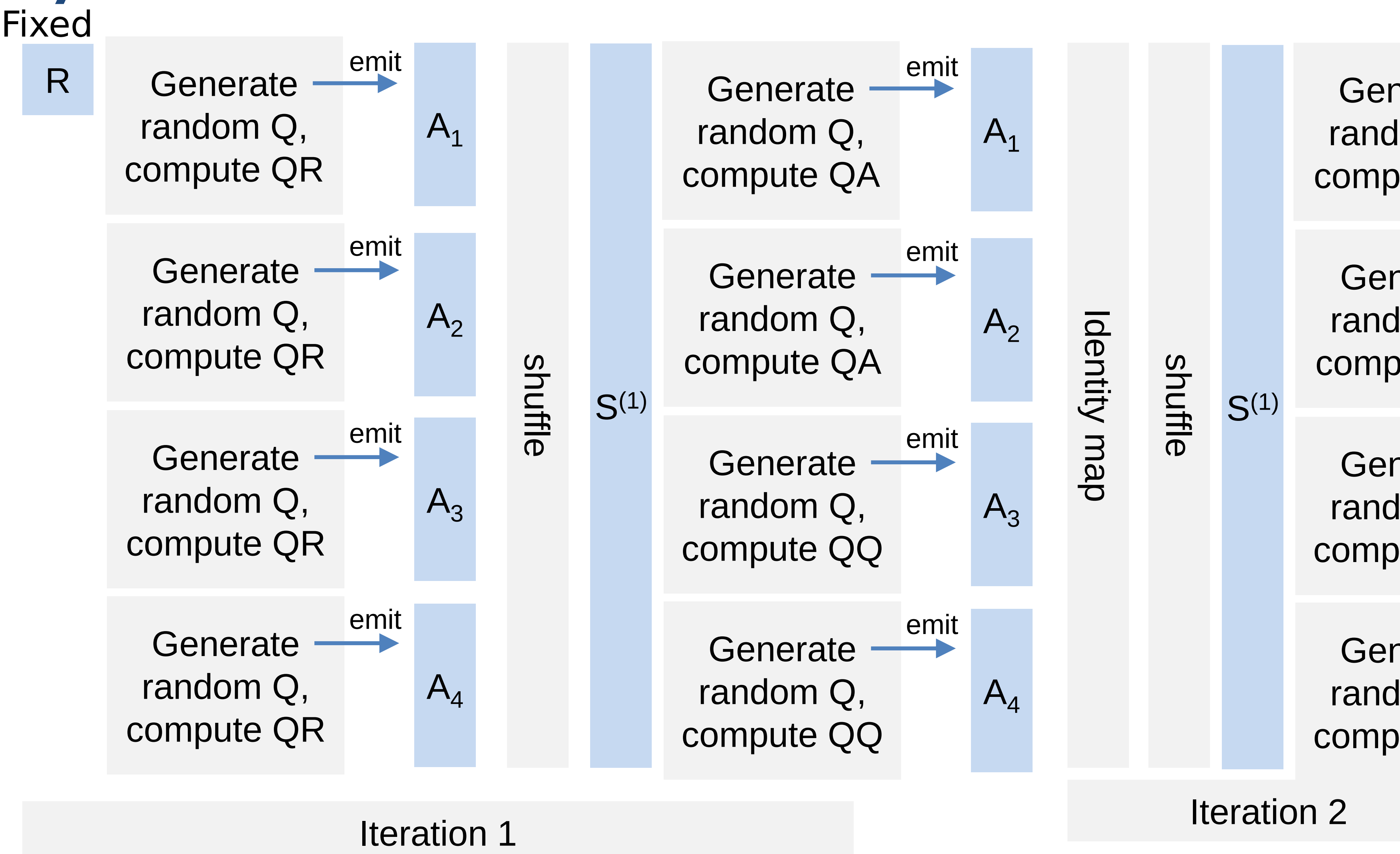
We first solved a regression problem by trying to predict the sum of red-pixel values in each image as a linear combination of the gray values in each image. Formally, if r_i is the sum of the red components in all pixels of image i, and $G_{i,j}$ is the gray value of the jth pixel in image i, then we wanted to find $\min \sum_i (r_i - \sum_j G_{i,j} s_j)^2$. There is no particular importance to this regression problem, we use it merely as a demonstration.



Took about 30 minutes

Synthetic Problems

Synthetic


Fixed
R
Generate random Q, compute QR
emit
A1
A2
A3
A4
shuffle
S(1)
Generate random Q, compute QA
Generate random Q, compute QQ
Identity map
Iteration 1
Iteration 2

Varying Blocksize Synthetic Data

Cols.	Blks.	Iter. 1 Maps	Secs.	Iter. 2 Secs.
50	2	8000	424	21
—	3	—	399	19
—	5	—	408	19
—	10	—	401	19
—	20	—	396	20
—	50	—	406	18
—	**100**	**—**	**380**	**19**
—	200	—	395	19
100	2	7000	410	21
—	3	—	384	21
—	5	—	390	22
—	**10**	**—**	**372**	**22**
—	20	—	374	22
1000	2	6000	493	199
—	3	—	432	169
—	**5**	**—**	**422**	**154**
—	10	—	430	202
—	20	—	434	202

Using the C++ framework

Varying splitsize *Synthetic Data*

Cols.	Iters.	Split (MB)	Maps	Secs.
50	1	64	8000	388
—	—	256	2000	184
—	—	512	1000	149
—	2	64	8000	425
—	—	256	2000	220
—	—	512	1000	191
1000	1	512	1000	666
—	2	64	6000	590
—	—	256	2000	432
—	—	512	1000	337

Using the C++ framework

Comparing Frameworks Synthetic Data

	Iter 1				
	Map		Red.		Total
	Secs.	QR (s.)	Secs.	QR (s.)	Secs.
Dumbo	911	67725	884	2160	960
hadoopy	581	70909	565	2263	612
C++	326	15809	328	485	350
Java-MTJ					436

	Iter 2				Overall
	Map	Red.		Total	
	Secs.	Secs.	QR (s.)	Secs.	Secs.
Dumbo	5	214	80	217	1177
hadoopy	5	112	81	118	730
C++	5	34	15	37	387
Java-MTJ				66	502

All values from Hadoop Job tracker. 100,000,000-by-500 matrix.