Matrix Math at Scale with Apache Mahout and Spark

Andrew Musselman
akm@apache.org

About Me

Professional

Data science and engineering, Chief Analytics Officer at A2Go

Software engineering, web dev, data science at online companies

Chair of Mahout PMC; started on Mahout project with a bug in the *k*-means method

Personal

Live in Seattle

Two decent kids, beautiful and supportive photographer wife

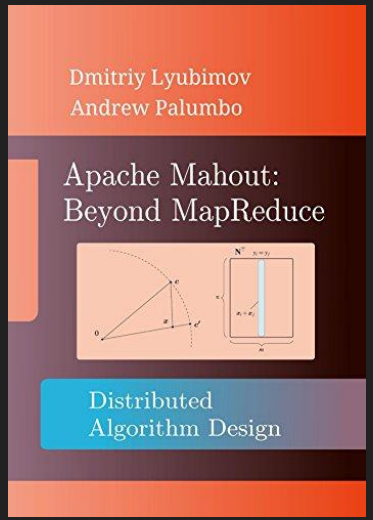
Snowboarding, bicycling, music, sailing, amateur radio (KI7KQA)

Co-host of podcast Adversarial Learning with @joelgrus

Recent Publications on Mahout

Apache Mahout: Beyond MapReduce

Dmitriy Lyubimov and Andrew Palumbo



https://www.amazon.com/dp/B01BXW0HRY

Encyclopedia of Big Data Technologies

Apache Mahout chapter by A. Musselman

https://www.springer.com/us/book/9783319775241

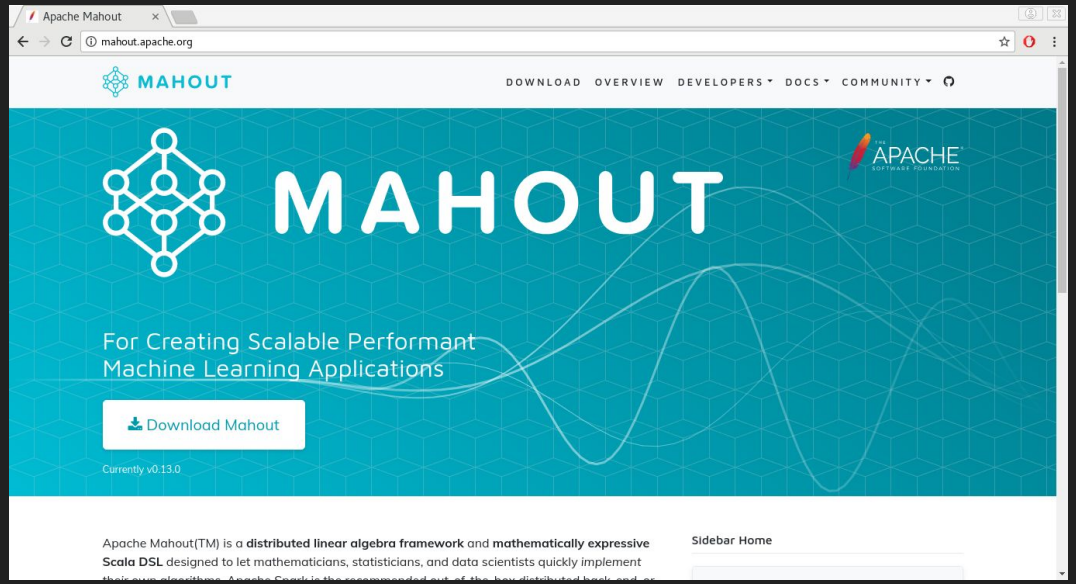
Apache Mahout Web Site Relaunch

http://mahout.apache.org

Thanks to Dustin VanStee, Trevor Grant, and David Miller (https://startbootstrap.com)

Jekyll-based, publish with push to source control repo

RIP Little Blue Man



Getting Started with Apache Mahout

- Project site at http://mahout.apache.org
- Mahout channel on The ASF Slack domain
  - #mahout on https://the-asf.slack.com
- Mailing lists
  - User and Dev lists
  - https://mahout.apache.org/general/mailing-lists,-irc-and-archives.html
- Clone the source code
  - https://github.com/apache/mahout
- Or get a pre-built binary build
  - “Download Mahout” button on http://mahout.apache.org
- Small, responsive and dedicated project team
- Experiment and get as close to the underlying arithmetic as you want to

Agenda

- Intro/Motivation
- Samsara DSL and Syntax
- Matrix Multiplication Optimizations
- JVM/ViennaCL/CUDA
- Install Mahout/Spark
- The REPL
- Other New Stuff: Zeppelin, Algorithm Development Framework
- Next Steps/Conclusion

Intro/Motivation

Intro

About Apache Mahout

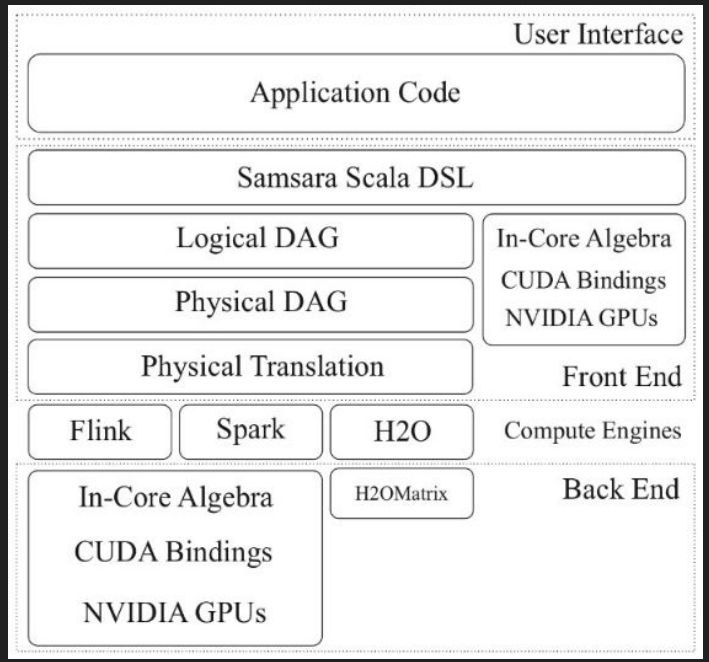
- Distributed linear algebra framework running on Spark, Flink, H2O
- Mathematically expressive Scala DSL
- Pluggable compute back-end (Spark recommended, Flink supported)
- Modular native solvers for CPU/GPU/CUDA acceleration
- Designed for fast experimentation with clean, math-like syntax
- Prototype to production with the same code

About Apache Spark

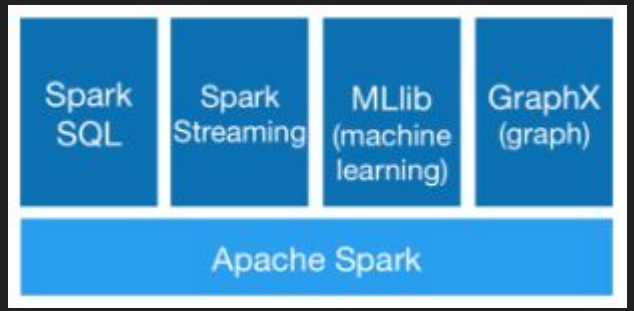
- Scalable distributed data processing and analytics engine
- Solid replacement for Hadoop MapReduce-based processes
- Cached results between steps eliminates re-scanning large files
- Scala, Python, R, SQL APIs
- MLLib machine learning library
- GraphX graph processing library

Intro

Mahout Architecture



Spark Architecture



Motivation: Why Matrix Math?

Machine learning foundations in vectors and matrices, arithmetic

Example data sets and corresponding vectors/matrices:

- Website access logs: vectors are visitors identified by user or cookie ids, and values are # of times visiting any given product page
- Banking transactions: vectors are customer ids or account numbers, values are transaction amounts for each vendor id
- Oil well drilling site sensor data: vectors are equipment ids, with values being reported value of each sensor on the equipment at any given timestamp
- Movie ratings: vectors are user ids, and values are 1-5 “star” rating for each movie

Motivation: Why Matrix Math?

Typical requirements of a machine learning method:

- Highly iterative
- Large-scale data sets

Around version 0.10 of Mahout it became obvious that using Hadoop MapReduce was causing more pain than it was solving, due to massively redundant data reads required

Motivation: Why Not Python/R?

Scale issues

Data set size

Number of iterations

Run-time expensive or impossible

Frameworks/products to parallelize/distribute compute are out there but are maturing or incomplete, e.g., Dask for Python, Revolution for R

Motivation: Why Not Just Use Spark MLLib?

Unique Spark and Scala idioms required

Skill and experience with these idioms needed

Translating symbolic math to code time-consuming and error-prone

Motivation: Samsara DSL/Syntax Bridging the Gap

Math-like idioms and flavor

Scalability built-in

Templating for algorithm development

Simpler translation from machine learning papers to code

Samsara DSL and Syntax

Samsara DSL and Syntax

Samsara *A'A*

```
val C = A.t %*% A
```

MLLib *A'A*

```
val C = A.transpose().multiply(A)
```

Samsara DSL and Syntax

Computation in distributed stochastic PCA (dSPCA):

$$G = BB^T - C - C^T + \xi^T \xi s_q{}^T s_q.$$

In Samsara DSL:

```
val G = B %*% B.t - C - C.t + (xi dot xi) * (s_q cross s_q)
```

Samsara DSL and Syntax

To import DSL for in-core linear algebra (automatic in the REPL):

```
import org.apache.mahout.math._

import scalabindings._

import RLikeOps._
```

Instantiating Vectors

```
// Dense vectors:

val denseVec1: Vector = (1.0, 1.1, 1.2)

val denseVec2 = dvec(1, 0, 1, 1, 1, 2)


// Sparse vectors:

val sparseVec1: Vector = (5 -> 1.0) :: (10 -> 2.0) :: Nil

val sparseVec1 = svec((5 -> 1.0) :: (10 -> 2.0) :: Nil)
```

Instantiating Matrices

```
// Dense matrices:

val A = dense((1, 2, 3), (3, 4, 5))


// Sparse matrices:

val A = sparse(
  (1, 3) :: Nil,
  (0, 2) :: (1, 2.5) :: Nil
)
```

Some Special Matrix Inits

```
// Diagonal matrix with constant diagonal elements:

diag(3.5, 10)


// Diagonal matrix with main diagonal backed by a vector:

diagv((1, 2, 3, 4, 5))


// Identity matrix:

eye(10)
```

Arithmetic and Assignment

```
// Plus/minus:

a + b

a - b

a + 5.0

a - 5.0

// Hadamard (elementwise) product:

a * b

a * 0.5
```

```
// Operations with assignment:

a += b

a -= b

a += 5.0

a -= 5.0

a *= b

a *= 5
```

Other Operators

```
// Dot product:

a dot b

// Cross product:

a cross b


// Matrix multiply:

a %*% b
```

```
// Optimized right and left multiply
with a diagonal matrix:

diag(5, 5) :%*% b

A %*%: diag(5, 5)

// Second norm, of a vector or matrix:

a.norm

// Transpose:

val Mt = M.t
```

Decompositions

```
import org.apache.mahout.math.decompositions._

// Cholesky decomposition

val ch = chol(M)

// SVD

val (U, V, s) = svd(M)

// In-core SSVD

val (U, V, s) = ssvd(A, k = 50, p = 15, q = 1)
```

```
// EigenDecomposition

val (V, d) = eigen(M)

// QR decomposition

val (Q, R) = qr(M)
```

More Samsara Reference

https://mahout.apache.org/users/environment/in-core-reference.html

Matrix Multiplication Optimizations

Optimization of *A'A*

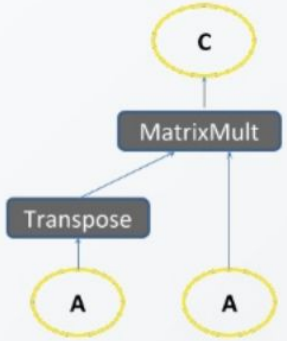
Example of an algebraic optimization

- **Mahout-Samsara computes C = A'A via row-outer-product formulation:** $A^T A = \sum_{i=0}^{m} a_i a_i^T$
- **Executes in a single pass over row-partitioned A**

- **Computation of A'A:**

```
val C = A.t %*% A
```

- **Naïve execution**
- **1st pass: transpose A (requires repartitioning of A)**
- **2nd pass: multiply result with A (expensive, potentially requires repartitioning again)**

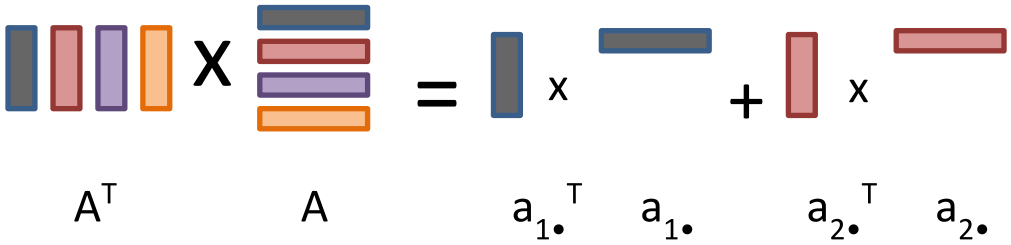


- **Logical optimization**
- **Optimizer rewrites plan to use logical operator for Transpose-Times-Self matrix multiplication**
- **Single pass: multiply partitioned rows by themselves as transposed columns**

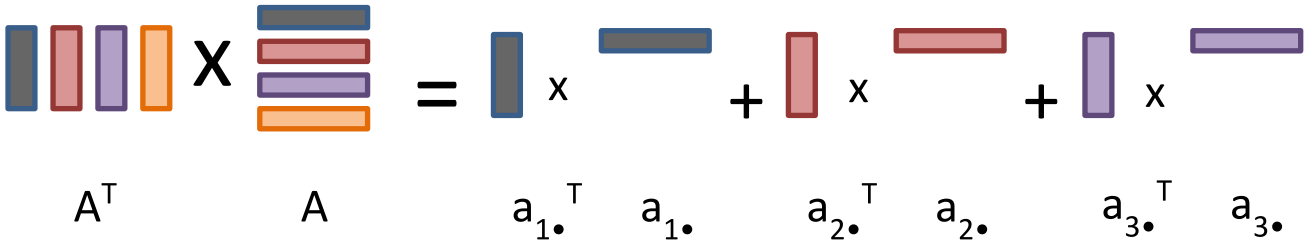
Optimization of *A'A*

Optimization of *A'A*

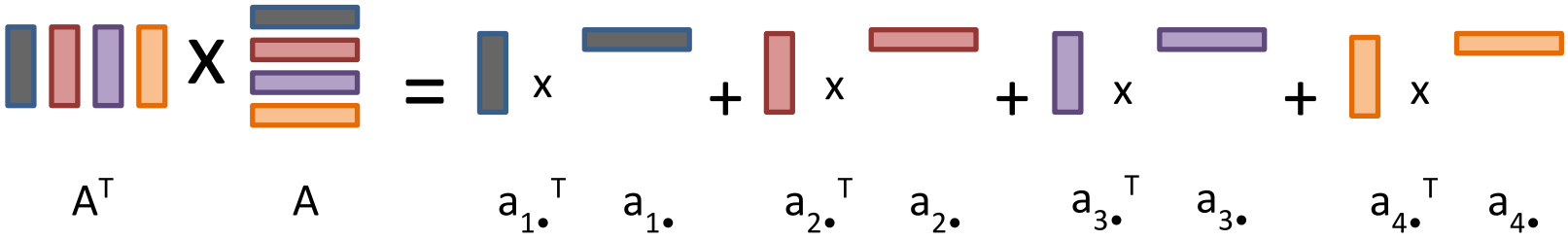
Optimization of *A'A*



Optimization of *A'A*



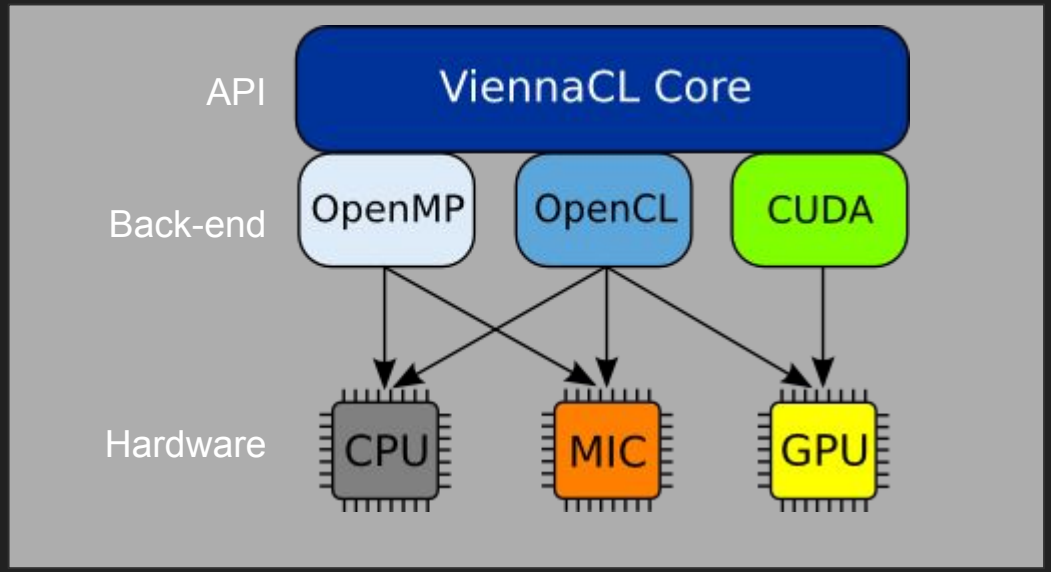
Optimization of *A'A*



JVM/ViennaCL/OpenMP/CUDA

Getting Outside the JVM

To do math outside the JVM Mahout uses ViennaCL as a facade layer in front of OpenMP (for multi-core CPU) and CUDA (for GPU) for computation



Install Mahout/Spark

Install Spark

Visit https://spark.apache.org/downloads.html, select Spark and Hadoop versions or directly download:

```
$ wget https://archive.apache.org/dist/spark/spark-2.1.1/spark-2.1.1-bin-hadoop2.7.tgz

$ tar xzvf spark-2.1.1-bin-hadoop2.7.tgz

$ ./spark-2.1.1-bin-hadoop2.7/sbin/start-all.sh

$ export SPARK_HOME=$PWD/spark-2.1.1-bin-hadoop2.7
```

Visit http://localhost:8080, get Spark Master URL, e.g., spark://bob:7077

```
$ export MASTER=spark://localhost:7077
```

Install Mahout Binary

Visit http://mahout.apache.org/general/downloads, click “Download Mahout,” or

```
$ wget http://apache.cs.utah.edu/mahout/0.13.0/apache-mahout-distribution-0.13.0.tar.gz

$ tar xzvf apache-mahout-distribution-0.13.0.tar.gz

$ export MAHOUT_HOME=$PWD/apache-mahout-distribution-0.13.0

$ cd apache-mahout-distribution-0.13.0

$ ./bin/mahout spark-shell
```

Install Mahout with Vienna/OMP/CUDA Support

Visit http://mahout.apache.org/general/downloads, go to “Download Latest,” or

```
$ wget
http://apache.cs.utah.edu/mahout/0.13.0/apache-mahout-distribution-0.13.0-src.tar.gz

$ tar xzvf apache-mahout-distribution-0.13.0-src.tar.gz

$ export MAHOUT_HOME=$PWD/apache-mahout-distribution-0.13.0

$ cd apache-mahout-distribution-0.13.0

$ mvn clean install -Pviennacl -DskipTests=true

$ ./bin/mahout spark-shell
```

The REPL

Playing with the Shell

Installation instructions and sample script:

https://github.com/andrewmusselman/talks/tree/master/open_source_summit

From http://mahout.apache.org/docs/latest/tutorials/samsara/play-with-shell.html

```
$ ./bin/mahout spark-shell
```

Linear Regression Example

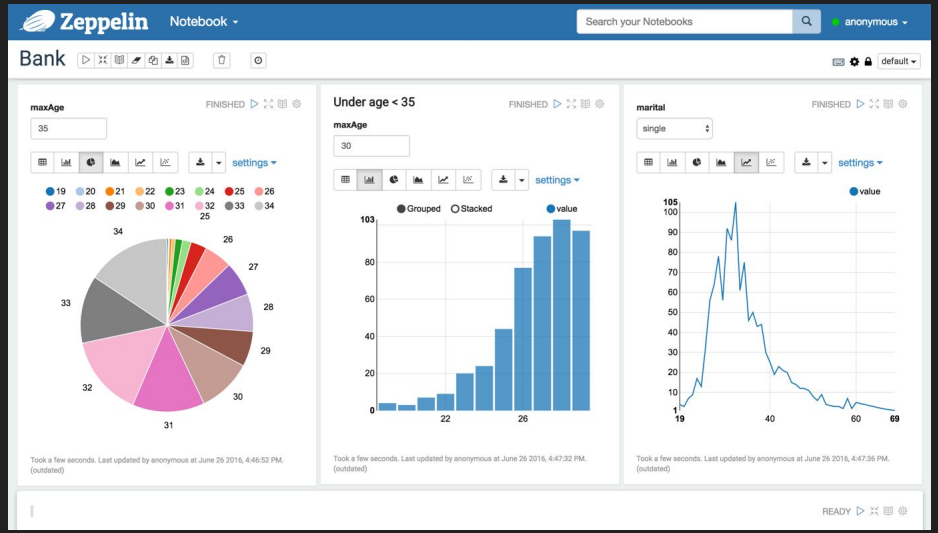
```
import org.apache.mahout.math._

val drmData = drmParallelize(dense(
  (2, 2, 10.5, 10, 29.509541),  // Apple Cinnamon Cheerios
  (1, 2, 12,   12, 18.042851),  // Cap'n'Crunch
  (1, 1, 12,   13, 22.736446),  // Cocoa Puffs
  (2, 1, 11,   13, 32.207582),  // Froot Loops
  (1, 2, 12,   11, 21.871292),  // Honey Graham Ohs
  (2, 1, 16,   8,  36.187559),  // Wheaties Honey Gold
  (6, 2, 17,   1,  50.764999),  // Cheerios
  (3, 2, 13,   7,  40.400208),  // Clusters
  (3, 3, 13,   4,  45.811716)), // Great Grains Pecan
  numPartitions = 2);

// Linear regression described by = Xb + e

// Extract feature matrix X and target variables y
val drmX = drmData(::, 0 until 4)
val y = drmData.collect(::, 4)

// Estimate beta with b ~ (X.t %*% X)^(-1) %*% X.t %*% y
val drmXtX = drmX.t %*% drmX
val drmXty = drmX.t %*% y
val XtX = drmXtX.collect
val Xty = drmXty.collect(::, 0)
val beta = solve(XtX, Xty)
```

```
def ols(drmX: DrmLike[Int], y: Vector) =
  solve(drmX.t %*% drmX, drmX.t %*% y)(::, 0)

def goodnessOfFit(drmX: DrmLike[Int], beta: Vector, y: Vector) = {
  val fittedY = (drmX %*% beta).collect(::, 0)
  (y - fittedY).norm(2)
}

// Add a bias variable
val drmXwithBiasColumn = drmX cbind 1
val betaWithBiasTerm = ols(drmXwithBiasColumn, y)
goodnessOfFit(drmXwithBiasColumn, betaWithBiasTerm, y)

// Faster with cached results
val cachedDrmX = drmXwithBiasColumn.checkpoint()
val cachedBetaWithBiasTerm = ols(cachedDrmX, y)
val goodness = goodnessOfFit(cachedDrmX, cachedBetaWithBiasTerm, y)
cachedDrmX.uncache()
```

Other New Stuff

Zeppelin and Algo Dev Framework

- Interpreter for Mahout in Zeppelin lets you work in **notebooks**!
  - https://mahout.apache.org/docs/latest/tutorials/misc/mahout-in-zeppelin



- Algorithm development framework standardizes methods needed for analytics jobs
  - http://mahout.apache.org/docs/latest/tutorials/misc/contributing-algos

Algorithm Development Framework

- Patterned after R and Python (sk-learn) APIs
- `Fitter` populates a `Model`
- `Model` contains parameter estimates, fit statistics, a summary, and a `predict()` method

```
class Foo[K] extends RegressorFitter[K] {

  def fit(drmX: DrmLike[K],
          drmTarget: DrmLike[K],
          hyperparameters: (Symbol, Any)*): FooModel[K] = {
    /**
      * Normally this section would have more code
      *
      */
    var model = new FooModel[K]
    model.summary = "This model has been fit, etc."
    model
  }
}

class FooModel[K] extends RegressorModel[K] {

  def predict(drmPredictors: DrmLike[K]): DrmLike[K] = {
    drmPredictors.mapBlock(1) {
      case (keys, block: Matrix) => {
        var outputBlock = new DenseMatrix(block.nrow, 1)
        keys -> (outputBlock += 1.0)
      }
    }
  }
}
```

Next Steps/Conclusion

Next Steps for Mahout

- jCUDA work in a branch, in master soon
- Multi-GPU
- Optimizing where data lives and where compute takes place
- Spark 2.1 and Scala 2.11 support
- Release 0.14.0 planned for Fall 2018

- **Try it out, get in touch!**

Thank You

Q&A

@akm