Large-Scale Linear Algebra with R

Bryan W. Lewis, Paradigm4, Inc.

blewis@paradigm4.com
blewis@illposed.net

July 2012

Topics

1. Big matrix algebra
2. Big, pseudo-sparse matrices
3. Distributed parallel linear algebra with virtual associative matrices
4. SciDB distrubuted matrices with R

Bigmemory

"Big Matrices" are similar to R matrices.

```
> library("bigmemory")
> A=big.matrix(5,5,type="double",backingfile="A")
> A[1:5,3]=rnorm(5)
> A[,]
     [,1] [,2]        [,3] [,4] [,5]
[1,]    0    0 -0.32619460    0    0
[2,]    0    0  0.82287399    0    0
[3,]    0    0 -1.71091496    0    0
[4,]    0    0 -0.06860057    0    0
[5,]    0    0 -0.19677675    0    0
```

Big Matrices

But, big matrices are different from standard R matrices:

- Their data may reside in files, allowing them to be larger than available system RAM.
- They are indexed by double-precision numbers (integers).
- Backing file data coforms to the in-memory representation.
- Some things don't work, and some others are provided by bigmemory-specific functions.

```
> svd(A)
Error in as.vector(data) :
  no method for coercing this S4 class to a vector
```

Bigalgebra

The “bigalgebra” package equips big matrices with arithmetic using standard (32- or 64-bit) BLAS libraries.

```
> library("bigalgebra")
> A=big.matrix(5,5,type="double",backingfile="A")
> A[,] = 1:25
> (A + A)[,]
     [,1] [,2] [,3] [,4] [,5]
[1,]    2   12   22   32   42
[2,]    4   14   24   34   44
[3,]    6   16   26   36   46
[4,]    8   18   28   38   48
[5,]   10   20   30   40   50
```

BLAS

The bigalgebra package includes the reference 64-bit BLAS.

It is easy to adapt the package to high-performance libraries like Intel MKL, AMD ACL, ATLAS and Goto blas.

The bigalgebra package even builds on Windows!

BLAS

The table shows average wall-clock times to compute `A %*% B` using the Intel MKL library, where `A` is 50,000 $\times$ 50,000 and `B` is 50,000 $\times$ 10.

Task	Average time in seconds
Fill with `rnorm` data	1045
Matrix product	3.2

All timings shown today used Amazon EC2 CC2 instances with 23 GB RAM (yes, 23) and two Intel Xeon X5570 2.9 GHz CPUs (eight physical CPU cores). Amazon EBS storage was used for files, which is kind of slow.

Bigalgebra

You can do a lot with arithmetic.

The “IRLBA” package efficiently computes partial singular value decompositions and can compute PCA. It only requires matrix multiplications that look like:

$$A \cdot B, \quad \text{and,} \quad A^T \cdot B,$$

where the matrix B typically has only a small number of columns compared to A.

Take advantage of the identity $(B^T A)^T = A^T B$ to avoid explicitly using the transpose of the big matrix (which could be expensive to access if A is really big).

IRLBA

IRLBA includes a user-defined matrix multiplication function argument, so it's easy to use with bigalgebra.

Here you go:

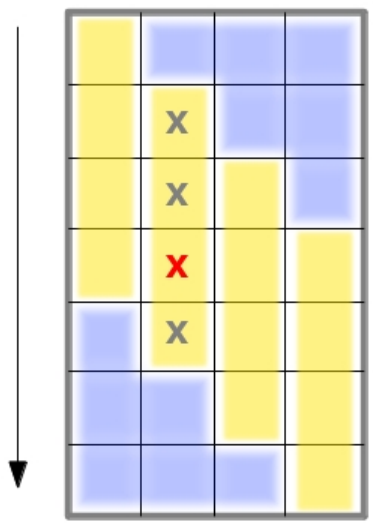
Truncated SVD

```
matmul = function(A, x, transpose=FALSE)
{
  if(transpose) {
    c = matrix(0,dim(t(x))[1],dim(A)[2])
    t( (t(x) %*% A)[,] )
  }
  c = matrix(0,dim(A)[1],dim(x)[2])
  (A %*% x)[,]
}
> dim(A)
[1] 50000 50000
> t1 = proc.time()
> S = irlba(A,nu=2,nv=2,matmul=f)
> proc.time() - t1

    user   system  elapsed
1846.223    0.877 1838.162
```

Pseudo-sparsity in big matrices

Big matrix file backings are sparse, in the file-system sense, if the file system supports it.

Big matrices exhibit pseudo-sparse behavior at the matrix level, writing data in page-sized chunks. Storing a single matrix element consumes a maximum of the system page size, usually 4k.



A single element (red X) will consume as much space as a page.

Example

```
> A=big.matrix(100000,100000,type="double",backingfile="A
    ")
> system("ls -lh A")
-rw-rw-r-- 1 ec2-user ec2-user 75G Jul 12 14:20 /tmp/A
```

What's going on here? Oh right, it's sparse:

```
> system("du -h /tmp/A")
0       /tmp/A

> A[55555,99999] = pi
> system("du -h A")
4.0K    A
```

Matrix and big matrix comparisons

Compute the product $A \cdot B$, where A is a 32,000$\times$32,000 sparse matrix and B is a 32,000$\times$100 dense matrix.

Class	Fill (s)	Product (s)	Object Size (b)
10% Fill-in by columns			
Big matrix	3.9	3.4	835067904
Matrix	26	16	1228929424
10% Fill-in by rows			
Big matrix	4.9	5.1	8208039936
Matrix	22	37	1228929424
1% Fill-in by columns			
Big matrix	0.03	3.2	83988480
Matrix	0.28	0.03	123009424

Some issues

It's not easy to run distributed parallel computations with big matrices.

Disk-backed big matrices slow down a lot for random-access problems that exceed memory size.

Virtual associative matrices

- Network instrumentation for bigmemory matrices
- Centralized Redis metadata repository
- Peer to peer data exchange
- Simple coordination semantics
- Intuitive feel

Instrumenting a big matrix:

```
> library('VAM')
> A1 = big.matrix(nrow=5,ncol=5, type='double',
    backingfile="A1")
> vnew(A1)
```

Explicitly reading data

The big matrix “A1” is now available from any R session on the network.

```
> vrd("A1",ridx=1:5,cidx=1:5)
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.1083507 0.1695219 0.5591294 0.5957889 0.6016608
[2,] 0.6655480 0.4482507 0.3225358 0.8868069 0.1704479
[3,] 0.5544777 0.9185981 0.7023079 0.3733112 0.4283674
[4,] 0.6864146 0.4463390 0.7199347 0.6886326 0.1564188
[5,] 0.8610068 0.2578736 0.3880187 0.4643071 1.3845350
```

Simple coordination

The “vin” function locks an array for writing. It returns after any current reads complete.

```
> A1 = vin("A1")
```

Subsequent “vrd” requests will block until the network association is restored with “vout:”

```
> vout("A1", A1)
```

Virtual big matrices

The virtual big matrix class makes things easy.

```
> L = matrix(c("A1","A2","A3","A4"),nrow=2,byrow=TRUE)
> A = vam(L)
> A
A 7 x 10 virtual matrix with layout:
     [,1] [,2]
[1,] "A1" "A2"
[2,] "A3" "A4"

> A[1:3,2:6]
     [,1] [,2] [,3] [,4] [,5]
[1,]    6   11   16   21    0
[2,]    7   12   17   22    0
[3,]    8   13   18   23    0
```

Virtual big matrices

VAM supports blockwise-sparse layouts.

```
> A1 = big.matrix(nrow=2,ncol=2, type='double', backingfile="A1")
> A1[,] = 1
> A2 = big.matrix(nrow=2,ncol=2, type='double', backingfile="A2")
> A2[,] = 2
> vnew(A1)
> vnew(A2)
> L = matrix(c("A1","","","A2"),nrow=2,byrow=TRUE)
> A = vam(L)

> A[,]
     [,1] [,2] [,3] [,4]
[1,]    1    1    0    0
[2,]    1    1    0    0
[3,]    0    0    2    2
[4,]    0    0    2    2
```

Parallel matrix multiply

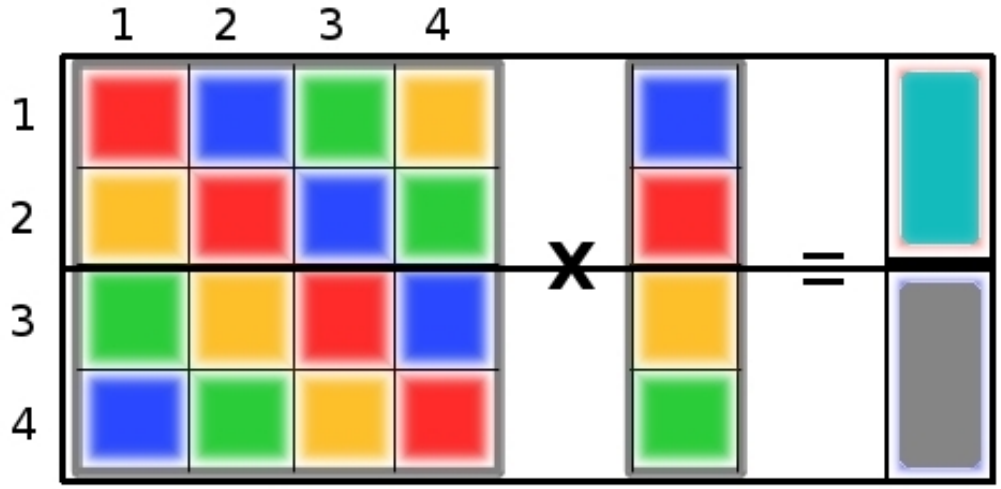
```
> library("doRedis")
> registerDoRedis(queue="example")

> L = foreach(j=1:2,.packages="VAM",.combine=c) %dopar%
+  {
+    key = paste("X",j,sep="")
+    ridx = ((j-1)*5 + 1):min((j*5),nrow(A))
+    X = A[ridx,] %*% B[,]
+    Y = as.big.matrix(X,backingfile=key)
+    vnew(Y, key)
+    key
+ }

> X = vam(matrix(L,nrow=2))
> sum(X[,] - A[,] %*% B[,])
[1] 0
```

Mix/match distribution and computation

16 blocks

4 instrumented big matrices (red, blue, green, yellow)

2 R compute workers (cyan, gray)



- Foreach and doRedis assign work on demand (pull).
- Data and computation grids may be coupled or not.
- (Worker) fault tolerant and elastic.
- Really good fit with EC2 and clouds.

SciDB and R

- SciDB is an open-source, array-oriented, parallel, distributed database.
- It knows about matrix arithmetic. And, we're about to introduce a Scalapack-based parallel linear algebra engine under the hood.
- Data distribution and parallelism are handled by SciDB automagically.
- Lots of extras: replication, failover, data versioning, easy deployment, SQL-like operations like joins, etc.
- We wrote a bigmemory-like interface between R and SciDB. It's one of the easiest ways to handle very large scale problems in R.

Truncated SVD, SciDB variant

```
matmul = function(A, x, transpose=FALSE)
{
  if(transpose) {
    b = t(x) %*% A; ans = t(b[,])
    scidbremove(b@name)
    ans
  }
  b = A %*% x; ans = b[,]
  scidbremove(b@name)
  ans
}
> A = scidb("A")
> dim(A)
[1] 50000 50000
> t1 = proc.time()
> S = irlba(A,nu=2,nv=2,matmul=f)
> proc.time() - t1

> proc.time()-t1
   user  system elapsed
 25.156  22.210 3944.247
```

Resources

- SciDB: http://scidb.org and http://paradigm4.com
- VAM is soon to be on r-forge: http://r-forge.r-project.org/projects/bigmemory/
- BigAlgebra is on R-forge. http://r-forge.r-project.org/projects/bigmemory/
- The irlba package is on CRAN.
- Bigmemory is on CRAN.