Adventures in Supercomputing with R

Lecture 9: Distributed Matrix Computation

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

2022/02/20 (updated: 2022-04-13)

Recall * SVD and Eigenvalue Decomposition Relationship

For an $n \times p$ matrix X and its SVD decomposition UDV^T, note eigenvalue decompositions:

$$X^T X = VDU^T UDV^T = VD^2 V^T$$

$$XX^T = UDV^T VDU^T = UD^2 U^T$$

So that if $n \ll p$ or $n \gg p$, we can take advantage of the shorter dimension in computing an SVD of X.

We compute the smaller crossproduct, do a small eigenvalure decomposition and recover the other singular vectors

$$U = XVD^{-1} \qquad \text{or} \qquad V = X^T UD^{-1}$$

with matrix multiplication.

*Lecture 7

This leads to a fast distributed SVD algorithm for skinny matrices

If $n \gg p$, we partition by blocks of rows, $X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_s \end{bmatrix}$, across s R-sessions,

then $A = X^T X = \sum_{i=1}^{s} X_i^T X_i$ and

```
A = allreduce(crossprod(X))
```

delivers the full crossproduct to all R-sessions, assuming p is small enough so that a $p \times p$ matrix fits on one node.
This offers a way to compute SVD on extremely large skinny matrices.

When $n \ll p$, we partition by blocks of columns and compute XX^T.

This leads to a fast distributed SVD algorithm for skinny matrices

Since A is on every processor, compute its eigenvalue decomposition $A = VD^2V^T$ and then

$$U = XVD^{-1} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix} VD^{-1} = \begin{bmatrix} X_1VD^{-1} \\ X_2VD^{-1} \\ \vdots \\ X_pVD^{-1} \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_p \end{bmatrix}$$

The result is V and D on every processor, while U is distributed the same way as X started.

Computing the eigenvalue decomposition of A on every processor is redundant, but it is faster than computing it on one processor and sending it to all.

Exercise: Show the algebra for the column partitioned case of $n \ll p$.

Now the R code

Starting with X distributed across processors by blocks of rows,

```
X = ## a function that reads a block of global X rows
A = allreduce(crossprod(X))
edA = eigen(A, symmetric = TRUE)
V = edA$vectors
d = sqrt(edA$values)
U = X %*% V %*% diag(1/d)
## or faster: U = sweep(X %*% V, 2, d, FUN = "\")
...
## use U and V in further operations
```

Is this relevant to our MNIST data and SVD model?

Our MNIST application, X has $n \gg p$. Our model (basis vectors) is the first k columns of V, which is $k \times p$, the shorter right singular vectors.

Following the preceding ideas, we could partition X by blocks of rows, compute $X^T X$ in parallel, and use the eigenvalue decomposition instead of SVD to obtain V.

We don't need U but given a new set of images Z to approximate (and classify), we can partition Z by blocks of rows and compute their approximations in parallel.

So this **is** *relevant to the MNIST SVD model, but Exercise08 asks to parallelize the independent crossvalidation instances. (Also the data is not large-enough to need this and we need to cover one more topic first: reading data in parallel.)*

Distributed QR decomposition

A similar approach can be used with QR decomposition.

Let $X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_s \end{bmatrix}$, with local QR decompositions $\begin{bmatrix} Q_1 R_1 \\ Q_2 R_2 \\ \vdots \\ Q_s R_s \end{bmatrix}$.

Here R_i are upper triangular and $Q_i^T Q_i = I$ and $Q_i Q_i^T = I$. $\qquad (^*)$

Suppose we perform another QR decomposition of $\begin{bmatrix} R_1 \\ R_2 \end{bmatrix} = Q_3 R_{12}$. Joining these operations, we have $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} Q_{31} \\ Q_{32} \end{bmatrix} R_{12}$.

Does the combined decomposition have the required properties $(^*)$?

Distributed QR decomposition

$$\begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} Q_{31} \\ Q_{32} \end{bmatrix} = \begin{bmatrix} Q_1 Q_{31} \\ Q_2 Q_{32} \end{bmatrix}$$

$$\begin{bmatrix} Q_1 Q_{31} \\ Q_2 Q_{32} \end{bmatrix}^T \begin{bmatrix} Q_1 Q_{31} \\ Q_2 Q_{32} \end{bmatrix} = [Q_{31}^T Q_1^T Q_1 Q_{31} + Q_{32}^T Q_2^T Q_2 Q_{32}] = Q_3^T Q_3 = I.$$

It also works for the outer crossproduct (Exercise).

Since it works for a pair of R factors, it works for combining all blocks.

Further, note that this is both commutative and associative.

The code for distributed QR

```
suppressMessages(library(cop))  # on GitHub at RBigData/cop
X = ## a function that reads a block of global X rows
R = qr_allreduce(X)
...
## use R for further operations in every rank
```

Allocating R instances to nodes

```
#!/bin/bash
#PBS -l select=number-nodes:mpiprocs=total-R-sessions
. . .
time mpirun --map-by ppr:per-node-R-sessions:node Rscript hello_balar
```

- `nummber-nodes`: nodes requested
- `total-R-sessions`: how many R sessions in all nodes
- `per-node-R-sessions`: how many R sessions in one node

For example, run on two nodes with 8 R sessions per node:

```
#!/bin/bash
#PBS -l select=2:mpiprocs=16
. . .
time mpirun --map-by ppr:8:node Rscript hello_balance.R
```

Running on Barbora

Barbora is up and running and they installed R the same way as on Karolina.

It has a separate file system so you will need to clone any files you need again.

Same scripts should work on Barbora as on Karolina, minding that Barbora nodes have 32 cores.

Collective operations must run on all ranks

- No nesting of collective operations
  - reduce() allreduce(), gather(), allgather(), comm.print(), comm.cat(), bcast(), barrier()
- Assign result first, then use in another operation
- Careful inside conditional operations
- A single rank not participating in a collective operation can hang the job

Collective operations must run on all ranks

- Example: Print minimum across all ranks

```
## This hangs!
comm.print(allreduce(my_x, op = "min"))
```

```
## This works!
global_min = allreduce(my_x, op = "min")
comm.print(global_min)
```

Collective operations must run on all ranks

- Example: Broadcast a value from rank 0 to all

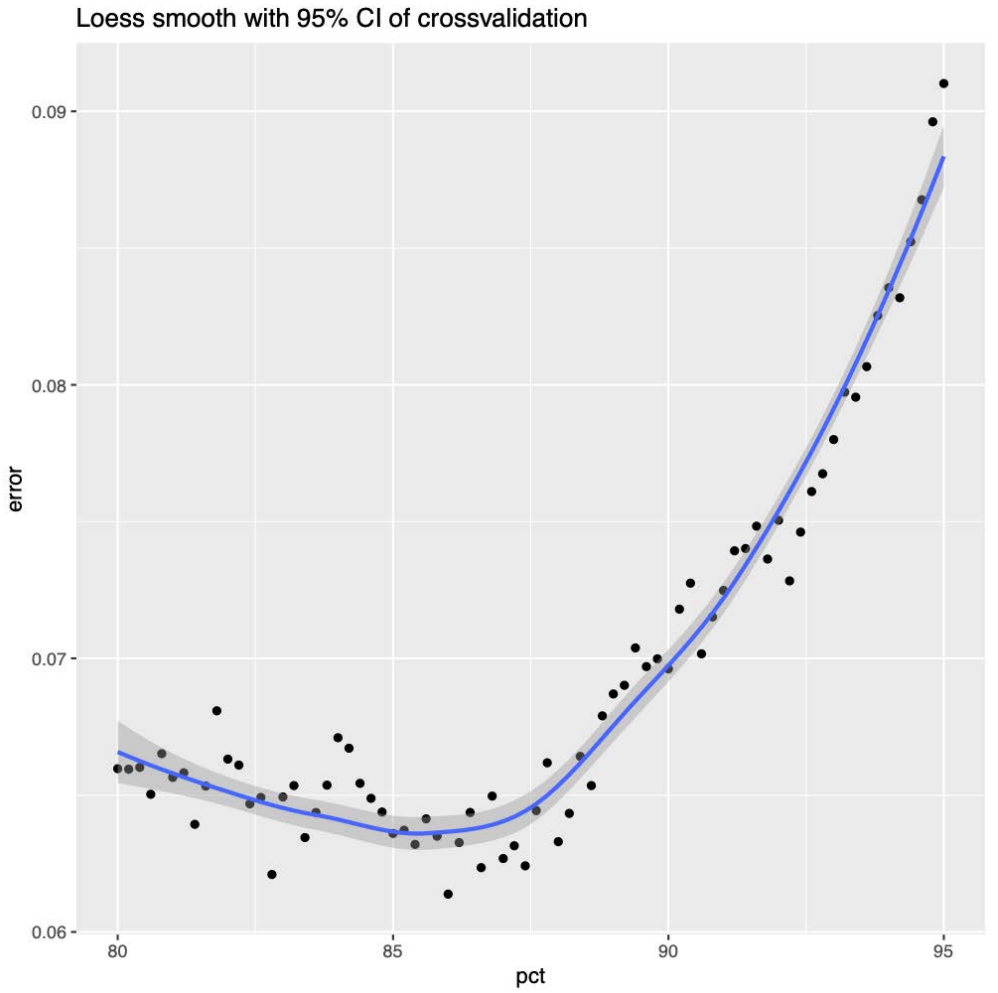
```
## This hangs
if(comm.rank() == 0) {
  x = some_function_of(my_data)
  bcast(x, rank.source = 0)
}
```

```
## This works
if(comm.rank() == 0) {
  x = some_function_of(my_data)
}  else {
  x = NULL
}
x = bcast(x, rank.source = 0)
```

Demo of simple MPI scripts ....

Directory `KPMS-IT4I-EX/mpi_scripts`

Update for Exercise 8: `pct = seq(85, 95, 0.2)`



How is your workflow?

- **ssh keys** enable fast workflow on unix and mac
  - git commit, **git push** in RStudio
    - code changes to GitHub
  - **git pull** on cluster
    - code changes to cluster
  - qsub on cluster (up arrow recalls previous commands)
    - submit run on cluster
  - **scp** on laptop (up arrow recalls previous commands)
    - get output to laptop (especially graphics)
- Help me understand WinSCP
  - Can you automate certain copies upon updates?