Adventures in Supercomputing with R

Lecture 12: Distributed Matrix, Randomized SVD

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

2022/02/20 (updated: 2022-05-04)

Construction of distributed matrix objects . . .

Fast randomized sketching algorithms (e.g. SVD) . . .

Discussion of group research projects . . .

Distributed Matrix Objects

- **shaq**: tall, skinny, dense matrix, row-block partition (package kazaam)
- **tshaq**: horizontal, skinny, dense matrix, column-block partition (package kazaam)
- **ddmatrix**: general, dense matrix, block-cyclic partition (package pbdDMAT)

Constructing a Distributed Matrix from Data

- MPI ranks read different data in contiguous blocks

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$a_{11}\ a_{12}\ a_{13}\ a_{21}\ a_{22}\ a_{23}\ a_{31}\ a_{32}\ a_{33}$

C, C++, NumPy **Row-Block**

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$a_{11}\ a_{21}\ a_{31}\ a_{12}\ a_{22}\ a_{32}\ a_{13}\ a_{23}\ a_{33}$

Fortran, R, Matlab **Column-Block**

- MPI ranks add attributes for global context

Three codes that follow are in the `KPMS-IT4I-EX/mpi` directory and run via submission script `mnist_ddemo.sh`:

```
#!/bin/bash
#PBS -N mnist_rsvd
#PBS -l select=1:mpiprocs=32
#PBS -l walltime=00:10:00
#PBS -q qexp
#PBS -e mnist_rsvd.e
#PBS -o mnist_rsvd.o

cd ~/KPMS-IT4I-EX/mpi
pwd

module load R
echo "loaded R"

## prevent warning when fork is used with MPI
export OMPI_MCA_mpi_warn_on_fork=0
export RDMAV_FORK_SAFE=1

## Fix for warnings from libfabric/1.12 bug
module swap libfabric/1.12.1-GCCcore-10.3.0 libfabric/1.13.2-GCCcore-

echo -e "\n>>>>>>>>> read and ddemo" >&2
time mpirun --map-by ppr:16:node Rscript mnist_ddemo.R
echo -e "\n>>>>>>>>> read and kazaam svd" >&2
time mpirun --map-by ppr:16:node Rscript mnist_kazaam.R
echo -e "\n>>>>>>>>> read and pbdML rsvd" >&2
time mpirun --map-by ppr:16:node Rscript mnist_rsvd.R
```

shaq and **ddmatrix** construction compared

```
# Compares data structures of shaq and ddmatrix with row-block input
source("mnist_read_mpi.R") # reads blocks of rows
suppressMessages(library(kazaam))

if(comm.rank() == 0) str(my_train) # local matrices
sq_train = kazaam::shaq(my_train) # shaq class distributed matrix fr
if(comm.rank() == 0) str(sq_train) # still local matrices but with g

sq_train2 = new("shaq", Data=my_train, nrows=nrow(my_train), ncols=nc
allreduce(all.equal(sq_train, sq_train2), op = "land")

suppressMessages(library(pbdDMAT))
init.grid()
bldim = c(allreduce(nrow(my_train), op = "max"), ncol(my_train))
gdim = c(allreduce(nrow(my_train), op = "sum"), ncol(my_train))
dmat_train = new("ddmatrix", Data = my_train, dim = gdim,
                 ldim = dim(my_train), bldim = bldim, ICTXT = 2)
comm.cat(comm.rank(), "dmat_train - Data dim:", dim(dmat_train@Data),
         "bldim:", dmat_train@bldim, "ICTXT:", dmat_train@ICTXT,
         "dim:", dmat_train@dim, "ldim:", dmat_train@ldim, "\n",
         all.rank = TRUE, quiet = TRUE)
comm.print(dmat_train)
cyclic_train = pbdDMAT::as.blockcyclic(dmat_train)
comm.print(cyclic_train)
comm.cat(comm.rank(), "cyclic_train - Data dim:", dim(cyclic_train@Da
         "bldim:", cyclic_train@bldim, "ICTXT:", cyclic_train@ICTXT,
         "dim:", cyclic_train@dim, "ldim:", cyclic_train@ldim, "\n",
         all.rank = TRUE, quiet = TRUE)
```

mnist_kazaam.R

```r
source("mnist_read_mpi.R") # reads blocks of rows
suppressMessages(library(kazaam))

## construct shaq matrix
sq_train = shaq(my_train)

## svd (shaq class: tall-skinny matrix)
options(warn = -1) ## suppress warnings about negative eigenvalues fo
train_svd = svd(sq_train, nu = 0, nv = 10)
comm.cat("kazaam top 10 singular values:", train_svd$d[1:10], "\n")

finalize()
```

mnist_rsvd.R

```r
source("mnist_read_mpi.R") # reads blocks of rows
suppressMessages(library(pbdDMAT))
suppressMessages(library(pbdML))
init.grid()

## construct block-cyclic ddmatrix
bldim = c(allreduce(nrow(my_train), op = "max"), ncol(my_train))
gdim = c(allreduce(nrow(my_train), op = "sum"), ncol(my_train))
dmat_train = new("ddmatrix", Data = my_train, dim = gdim,
                 ldim = dim(my_train), bldim = bldim, ICTXT = 2)
cyclic_train = as.blockcyclic(dmat_train)

rsvd_train = rsvd(cyclic_train, k = 10, q = 3, retu = FALSE, retv = F
comm.cat("rsvd top 10 singular values:", rsvd_train$d, "\n")

finalize()
```

Randomized sketching algorithms *, such as rsvd above, are fast new alternatives to classical numerical linear algebra computations. Guarantees are given with probability statements instead of classical error analysis.

* Martinsson, P., & Tropp, J. (2020). Randomized numerical linear algebra: Foundations and algorithms. Acta Numerica, 29, 403-572. https://doi.org/10.48550/arXiv.2002.01387

Randomized SVD via subspace embedding

Given an $n \times p$ matrix X and $k = r + 10$, where r is the *effective rank* of X:

1. Construct a $p \times l$ random matrix Ω
2. Form $Y = X\Omega$
3. Decompose $Y = QR$

Q is an orthogonal basis for the columnspace of Y, which with high probability is the columnspace of X. To get the SVD of X:

1. Compute $C = Q^T X$
2. Decompose $C = \hat{U}\Sigma V^T$
3. Compute $U = Q\hat{U}$
4. Truncate factorization to r columns

Randomized SVD via subspace embedding

Given an $n \times p$ matrix X and $k = r + 10$, where r is the *effective rank* of X:

1. Construct a $p \times l$ random matrix Ω
2. Let $Y_0 = \Omega$
3. For i in $1 : q$
    1. Decompose $Y_{i-1} = Q_i R_i$
    2. $Y_i = X(X^T Q_i)$
4. Decompose $Y_q = QR$

Q is an orthogonal basis for the columnspace of Y, which with high probability is the columnspace of X. To get the SVD of X:

1. Compute $C = Q^T X$
2. Decompose $C = \hat{U} \Sigma V^T$
3. Compute $U = Q\hat{U}$
4. Truncate factorization to r columns

Exercise 12

Produce a scaling graph for the `mnist_rsvd.R` code.

Optional: Use the `rsvd()` algorithm in the `mnist_svd_cv_mpi.R` code for the basis construction. Note: while you can nest *fork* parallelization inside *MPI*, you can not nest *MPI* inside *fork*. But you can nest OpenBLAS multithreading inside ScaLAPACK's MPI.

Discussion . . .