Adventures in Supercomputing with R

Lecture 10: Distributed Matrix Computation (continued)

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

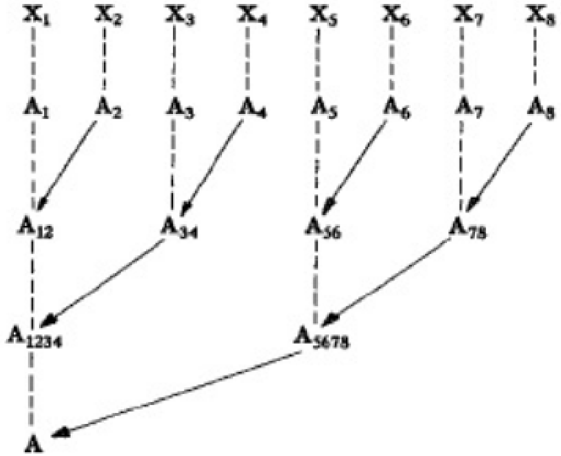
2022/02/20 (updated: 2022-04-20)

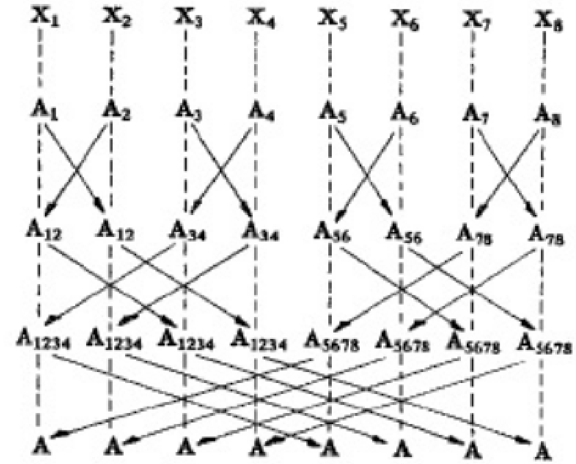
qr_allreduce.r

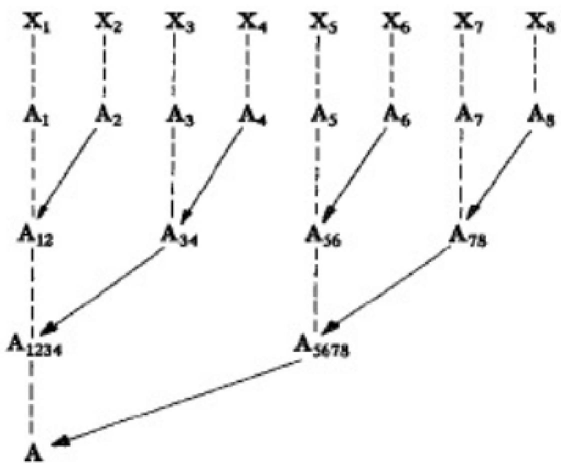
```r
library(cop, quiet = TRUE)

rank = comm.rank()
size = comm.size()

rows = 3
cols = 3
xb = matrix((1:(rows*cols*size))^2, ncol = cols) # a full matrix
xa = xb[(1:rows) + rank*rows, ]  # split by row blocks

comm.print(xa, all.rank = TRUE)
comm.print(xb)

## compute usual QR from full matrix
rb = qr.R(qr(xb))
comm.print(rb)

## compute QR from gathered local QRs
rloc = qr.R(qr(xa))  # compute local QRs
rra = allgather(rloc)  # gather them into a list
rra = do.call(rbind, rra)  # rbind list elements
comm.print(rra)  # print combined local QRs
ra = qr.R(qr(rra)) # QR the combined local QRs
comm.print(ra)

## use cop package to do it again via qr_allreduce
ra = qr_allreduce(xa)
comm.print(ra)
```

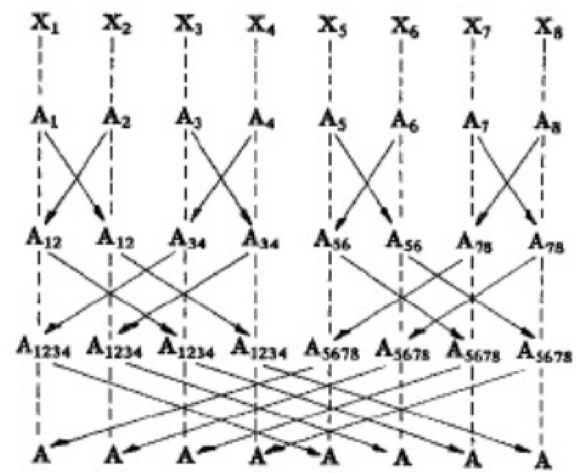
`qr_allreduce.r`

- `cop` version is faster because it does a reduce rather than a gather
- `gather` communication and memory grows and combined QR is serial
- `reduce` communication does not grow. reduce combines QRs in parallel. reduce
- Memory use scales with local size, not global size.
- QR is unique up to multiplication by a diagonal matrix of {+1, -1}


X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
A_{12} A_{34} A_{56} A_{78}
A_{1234} A_{5678}
A


X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
A_{12} A_{12} A_{34} A_{34} A_{56} A_{56} A_{78} A_{78}
A_{1234} A_{1234} A_{1234} A_{1234} A_{5678} A_{5678} A_{5678} A_{5678}
A A A A A A A A

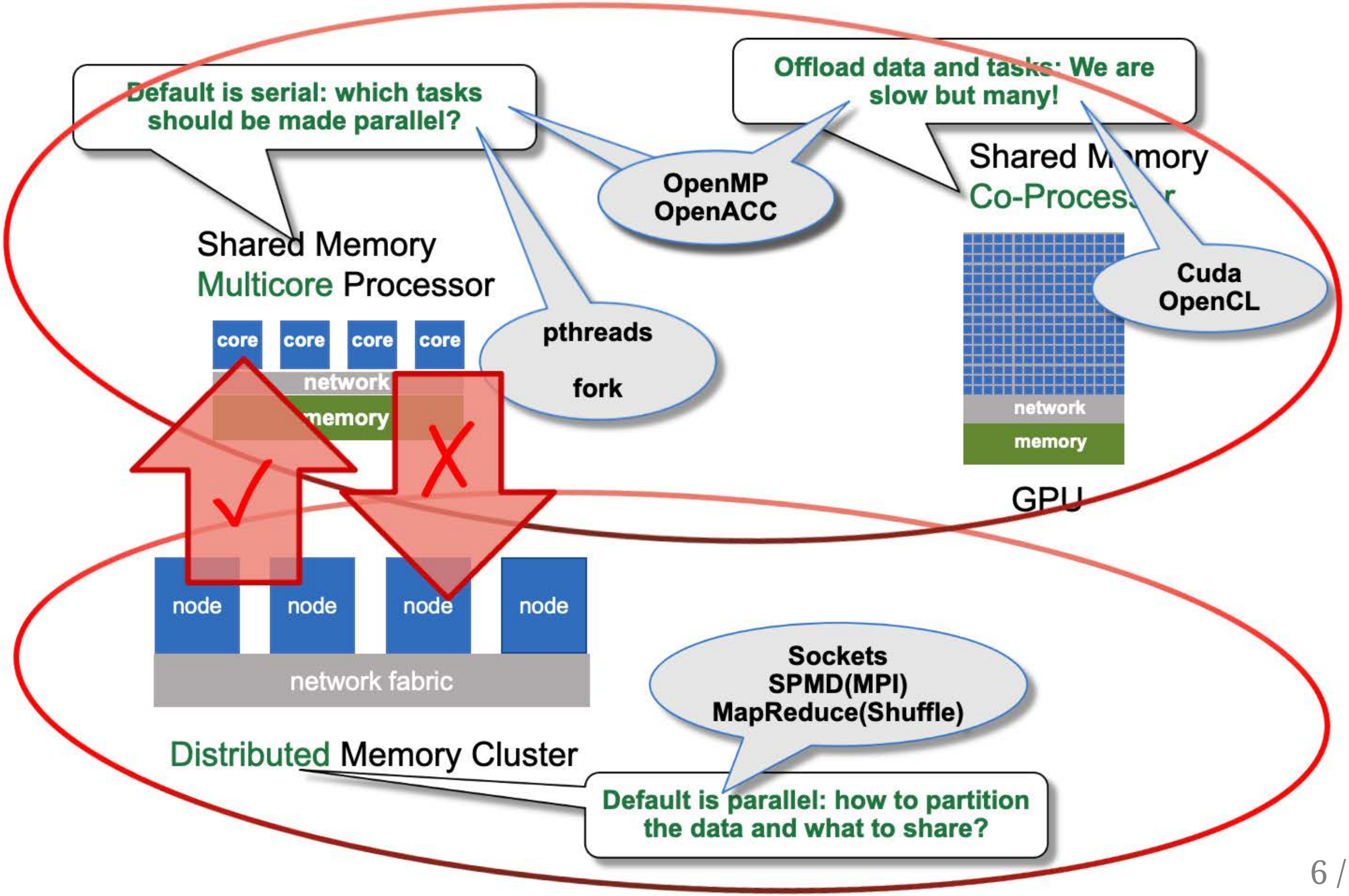

X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
A_{12} A_{34} A_{56} A_{78}
A_{1234} A_{5678}
A


X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
A_{12} A_{12} A_{34} A_{34} A_{56} A_{56} A_{78} A_{78}
A_{1234} A_{1234} A_{1234} A_{1234} A_{5678} A_{5678} A_{5678} A_{5678}
A A A A A A A A

mnist_read_mpi.r

```
suppressMessages( library( rhdf5 ) )
suppressMessages( library( pbdIO ) )

filename = "/scratch/project/dd-21-42/data/mnist/train.hdf5"
dataset1  = "image"
dataset2  = "label"

## get and broadcast dimensions to all processors
if ( comm.rank() == 0 ) {
   h5f = H5Fopen( filename, flags="H5F_ACC_RDONLY")
   h5d = H5Dopen( h5f, dataset1 )
   h5s = H5Dget_space( h5d )
   dims = H5Sget_simple_extent_dims( h5s )$size
   H5Dclose( h5d )
   H5Fclose( h5f )
} else dims = NA
dims = bcast( dims )

## get my local indices for contiguous data read
nlast = dims[length(dims)] # last dim moves slowest
my_ind = comm.chunk( nlast, form = "vector" )

## parallel read of local data
my_train = as.double( h5read( filename, dataset1, index = list( NULL,
my_train_lab = as.character( h5read( filename, dataset2, index = list
H5close( )

dim( my_train ) = c( prod( dims[-length(dims)] ), length(my_ind) )
my_train = t( my_train )  # now it's rowblock
```

Distributed Programming Works in Shared Memory



Mapping Threads to Cores

Theory and Reality

- Operating system manages core affinity
- Operating system tasks can compete
- Core switching occurs frequently
- MPI is more rigid and more planned

pbdR-ScaLAPACK-PBLAS-BLACS-MPI

- MPI: Message Passing Interface - *de facto* standard for distributed communication in supercomputing
  - Used for data mostly via collective communication - high level
  - `pbdMPI`, `kazaam`, and `cop` R packages
- ScaLAPACK: Scalable LAPACK - Distributed version of LAPACK (uses PBLAS/BLAS but not LAPACK)
  - 2d Block-Cyclic data layout - mostly automated in `pbdDMAT` package
  - BLACS: Communication collectives for distributed matrix computation
  - PBLAS: BLAS - distributed BLAS (uses shared memory BLAS within blocks)
  - `pbdDMAT` and `pbdML` R packages - most matrix operations identical to serial through overloading operators and `ddmatrix` class

Processor Grid and Complex Block Cyclic Layout

RBigData/pbdDMAT package on GitHub

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

(a) 1×6 (b) 2×3 (c) 3×2 (d) 6×1

Table: Processor Grid Shapes with 6 Processors

$$\left[\begin{array}{cc|cc|cc|cc|c}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\ \hline
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\ \hline
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\ \hline
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\ \hline
x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99}
\end{array}\right]_{9 \times 9}$$

$$\left[\begin{array}{cc|cc}
x_{11} & x_{12} & x_{17} & x_{18} \\
x_{21} & x_{22} & x_{27} & x_{28} \\ \hline
x_{51} & x_{52} & x_{57} & x_{58} \\
x_{61} & x_{62} & x_{67} & x_{68} \\ \hline
x_{91} & x_{92} & x_{97} & x_{98}
\end{array}\right]_{5 \times 4}
\left[\begin{array}{cc|c}
x_{13} & x_{14} & x_{19} \\
x_{23} & x_{24} & x_{29} \\ \hline
x_{53} & x_{54} & x_{59} \\
x_{63} & x_{64} & x_{69} \\ \hline
x_{93} & x_{94} & x_{99}
\end{array}\right]_{5 \times 3}
\left[\begin{array}{cc}
x_{15} & x_{16} \\
x_{25} & x_{26} \\ \hline
x_{55} & x_{56} \\
x_{65} & x_{66} \\ \hline
x_{95} & x_{96}
\end{array}\right]_{5 \times 2}$$

$$\left[\begin{array}{cc|cc}
x_{31} & x_{32} & x_{37} & x_{38} \\
x_{41} & x_{42} & x_{47} & x_{48} \\ \hline
x_{71} & x_{72} & x_{77} & x_{78} \\
x_{81} & x_{82} & x_{87} & x_{88}
\end{array}\right]_{4 \times 4}
\left[\begin{array}{cc|c}
x_{33} & x_{34} & x_{39} \\
x_{43} & x_{44} & x_{49} \\ \hline
x_{73} & x_{74} & x_{79} \\
x_{83} & x_{84} & x_{89}
\end{array}\right]_{4 \times 3}
\left[\begin{array}{cc}
x_{35} & x_{36} \\
x_{45} & x_{46} \\ \hline
x_{75} & x_{76} \\
x_{85} & x_{86}
\end{array}\right]_{4 \times 2}$$

```
> x <- as.rowblock( x )
> x <- as.blockcyclic( x )
> x <- redistribute( x, bldim=c(8, 8), ICTXT = 0 )
```

Exercise 8 ...

MPI version of crossvalidation for SVD basis functions

Added debugging strategies to Exercise 08 ...

More Debugging strategies

- Login with two terminal windows to cluster
  - Window1: submission, status, etc
    - Submit job
    - Check status, etc.
  - Window2: monitor with `top` on compute node
    - `check-pbs-jobs --check-all`
    - `ssh cn###`
    - `top`
- Multithreaded OpenBLAS show up as using more than 100% cpu
  - Example: with 4 threads, will show a little over 300%
- Fork with `mclappy` will show as separate R process
- MPI rank will show as separate R process

Only 4 weeks remain!

Remaining topics to learn

- More distributed matrix computation
  - `kazaam` and `pbdDMAT` packages
- Distributed I/O
  - csv files, hdf5 files, and others

Scaling your research on IT4I

- **Your group:** ~5 minute presentation of your problem that needs HPC
- **All:** Discuss how to do it
- See Exercise 10

The Power of Speed

- < 1 minute: Not getting bored
- < 5 seconds: Can optimize parameters
- < 1 second: can provide automated optimization
- It's not just memory of a larger system
- Speed can bring optimization and search options that are otherwise difficult or even unthinkable