Adventures in Supercomputing with R

Lecture 7: SVD and Friends, Begin Distributed

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

2022/02/20 (updated: 2022-03-30)

Exercise 4 Review ...

Exercise 6 Review ...

Truncated SVD as Regression Basis Vectors

Images as rows (observations)

Continuing with the MNIST data set of digit images from last time, here we construct A with each image as a row.

Let A be the matrix of n_A images of a single digit, the pixel values of each image as a row (an observation). We have image samples of each of 10 digits and we put them into separate matrices, like A, and consider use them for classification of new images.

The SVD of $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. If $\mathbf{u}_i$ and $\mathbf{v}_i$ are the columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, then

$$A = \sum_{i=1}^{p} d_i u_i v_i^T.$$

and image in row i is $\sum_{j=1}^{p}(d_j u_{ij})v_j$. The v_j are basis vectors constructed from the training data.

Truncated SVD as Regression Basis Vectors

From matrix approximation, we know that this SVD can be truncated to some $k \ll p$ components and still represent the matrix (and so the images) well.

The v_i are basis functions constructed from data. It is a set of orthogonalized "images", which are the regressors and the $d_i u_{ij}$ are the regression coefficients on the training data.

We can now look at classification of a new image of a digit by regressing it onto each of the 10 digit bases and classifying it into the category that fits best.

The tuning parameter k can be optimized with crossvalidation.

New Image Regression onto the Basis Vectors

Suppose we performed SVDs on the ten matrices A_d, each being a training set of a digit $d = 0, 1, \ldots, 9$, and produced a set of $p \times k$ matrices of k basis vectors $\tilde{V}_d$.

For a new image vector y, our digit d basis function model is $y = \tilde{V}_d \beta + \epsilon$. We drop the subscript d as we do the same for each digit.

Because the basis vectors are orthogonal, the normal equations $\tilde{V}^T \tilde{V} \hat{\beta} = \tilde{V}^T y$ are easily solved with $\hat{\beta} = \tilde{V}^T y$.

Note that because of the column truncation, we have $\tilde{V}^T \tilde{V} = I_k$ but $\tilde{V} \tilde{V}^T \neq I_p$. The L_2 loss is $\|y - \tilde{V} \tilde{V}^T y\|_2$.

For classification of the new image vector y, we choose the model with the smallest y regression model loss.

SVD and Eigenvalue Decompositions for Principal Components Analysis

In R, `princomp()` does an eigenvalue decomposition of the covariance (or correlation) matrix.

`prcomp()` computation proceeds via the singular value decomposition of the centered (and possibly scaled) data matrix.

$\mathrm{Cov}(X)$ centers the data $X_c = X - n^{-1}11^T X$ and returns $X_c^T X$.

For $X = UDV^T$, note that

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

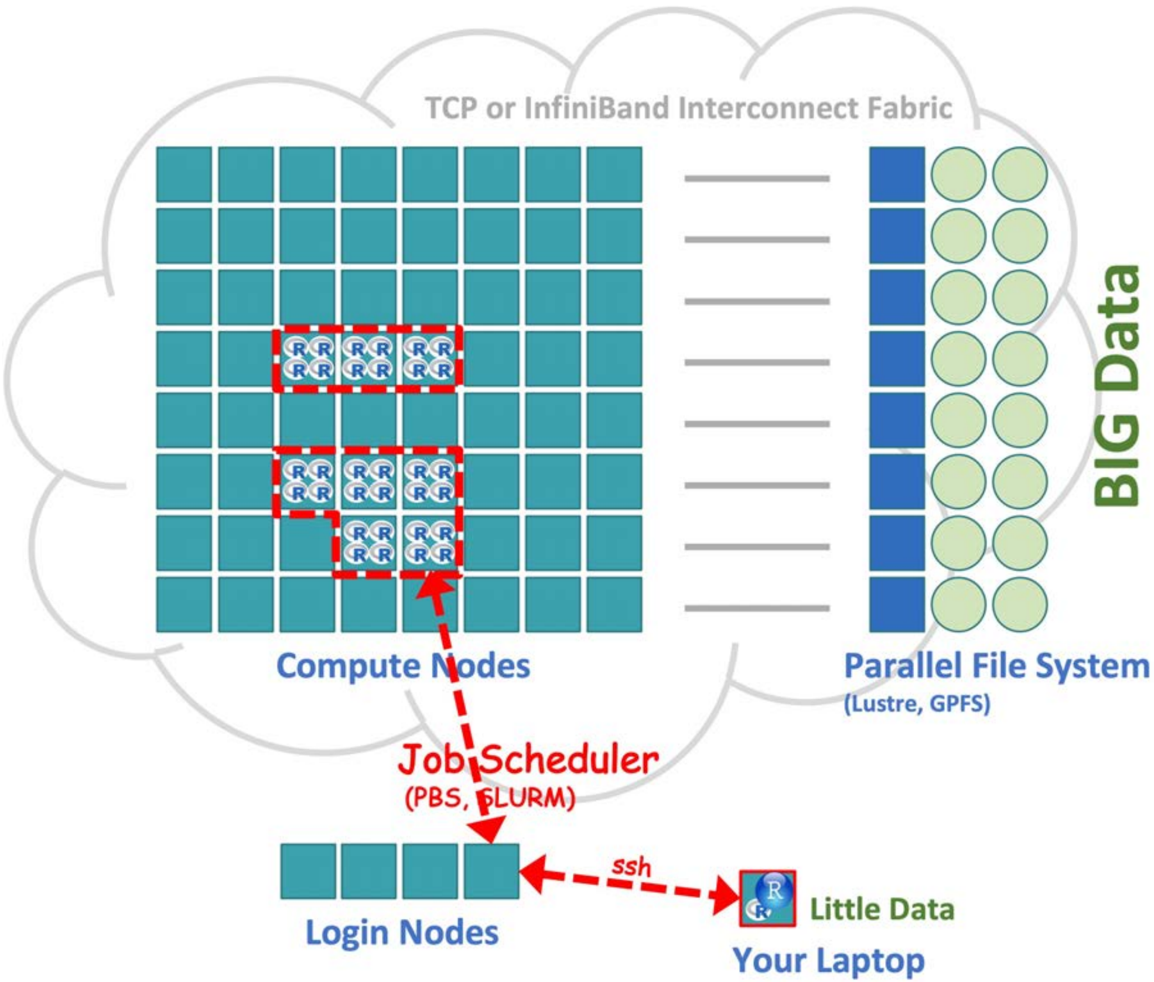
$$XX^T = UDV^T VDU^T = UD^2U^T$$

So that if $m \ll n$ or $m \gg n$ (a skinny X matrix), we can chose the smaller decomposition and recover the other $U = XVD^{-1}$ or $V = X^T UD^{-1}$ with matrix multiplication.

This results in easy fast distributed algorithms for PCA of skinny matrices as we will see in a future lecture.

Questions? … Demo …

Running Distributed on a Cluster



Parallel Computing Models

Shared memory parallel computing

- Processors have access to all memory
  - Locking mechanism
- Kinds: unix fork, pthreads, OpenMP, OpenACC
- Libraries: OpenBLAS, MKL, FlexiBLAS, PLASMA, MAGMA, etc.

Distributed memory parallel computing

- Processors have only local memory
  - Communication mechanism
- Kinds: MPI, MapReduce, DataFlow
- Libraries: OpenMPI, ScaLAPACK, PETSci, Trillinos, etc.

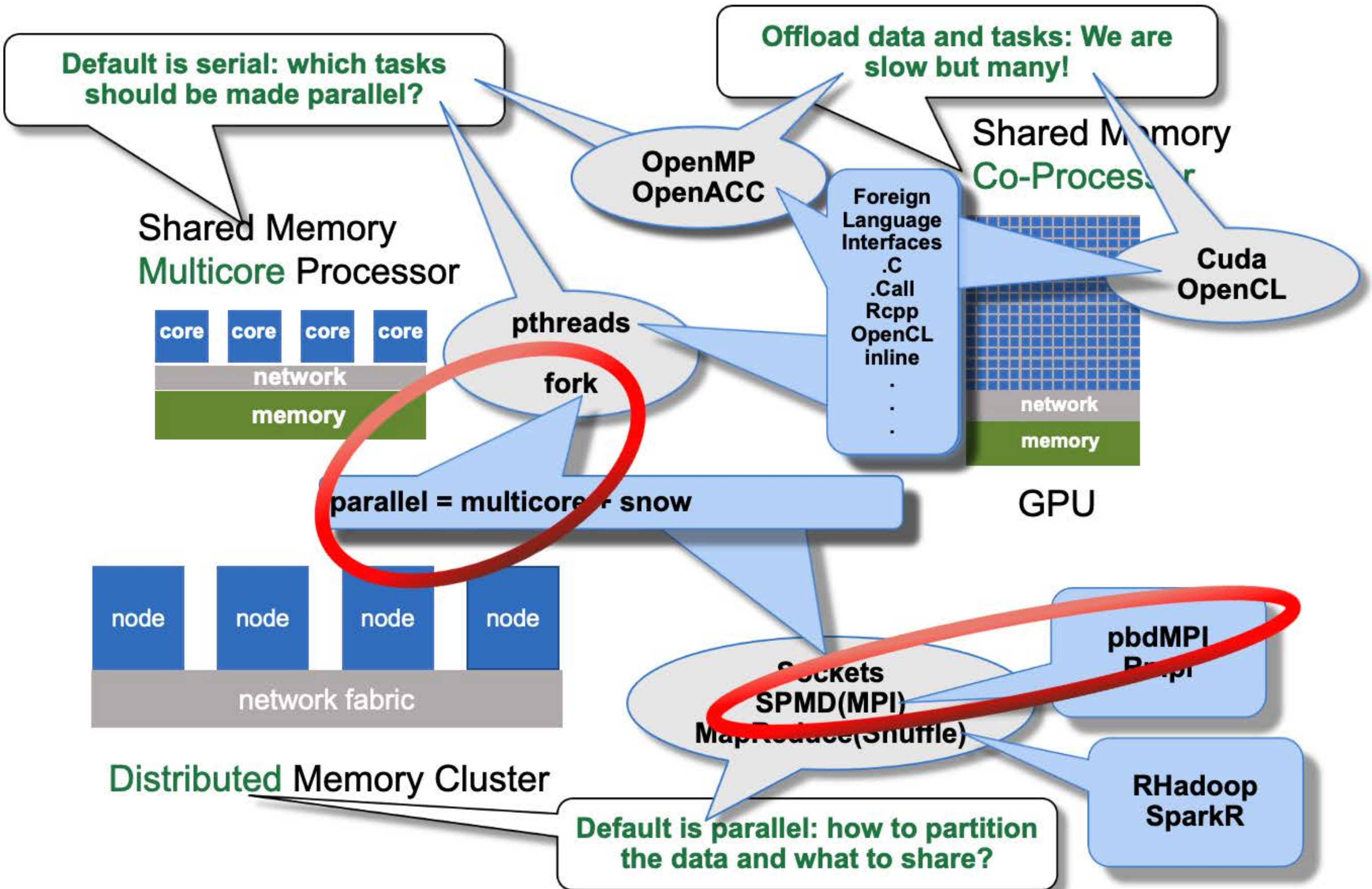
Distributed Programming Models

- Manager-workers
  - Common in simple cases (e.g. `mclapply()`)
- Single program, multiple data (SPMD)
  - Most common on supercomputer clusters
  - Most scalable because based on collaboration rather than control
  - Concept behind MPI design
- MapReduce
  - Common in data processing but falling out of favor because
  - Slow because includes all-to-all shuffle communication
- Dataflow
  - Dependency graph directed, still evolving
  - Sometimes combined with MPI for distributed control

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

R Interfaces to Low-Level Native Tools



Single Program Multiple Data (SPMD)

Hello world!

```
suppressMessages(library(pbdMPI))

msg = paste("Hello World! rank", comm.rank(), "out of", comm.size())
cat(msg, "\n")

finalize()
```

One code and a parallel mindset

A generalization of a serial code

Many rank-aware operations are automated

No manager, it is all cooperation

Explicit point-to-point communacations are an advanced topic

`pbdMPI`: reduce(X) allreduce(X)

Computing Crossproducts on 8 Processors of the First (1986) Intel IPSC/1 Hypercube

$$A = X^T X = \sum_i X_i^T X_i, \text{ where } X = \begin{bmatrix} X_1 \\ \vdots \\ X_8 \end{bmatrix}$$

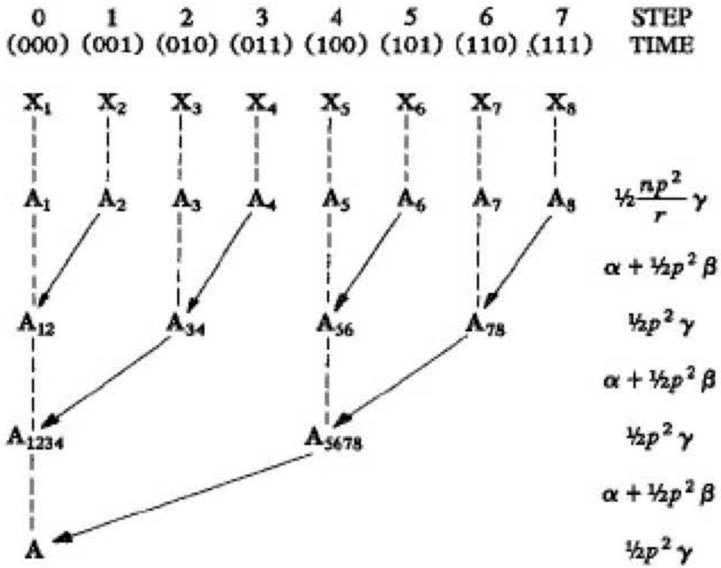


FIG. 4. *Computation of* **A** = **X'X** *on an 8-processor hypercube, with final result on processor 0.*

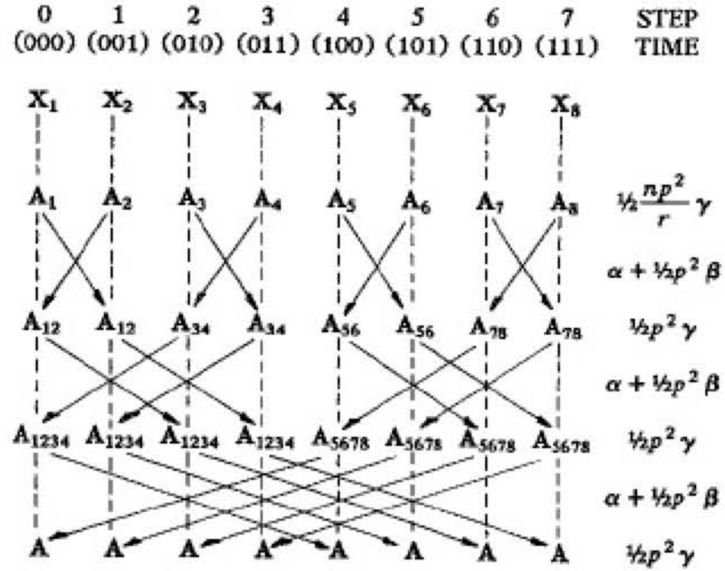


FIG. 6. *Computation of* **A** = **X'X** *on an 8-processor hypercube, with final result on all processors.*

Ostrouchov (1987). Parallel Computing on a Hypercube: An overview of the architecture and some applications. *Proceedings of the 19th Symposium on the Interface of Computer Science and Statistics*, p.27-32.