MPI Parallelization of Low Rank Matrices

Sameer Deshmukh

deshmukh.m.aa@m.titech.ac.jp

School of Computing, Tokyo Institute of Technology

Rio Yokota Lab

Abstract

One of the ways of computing the inverse of or solving very large dense matrices (more than a million by a million) is to calculate the LU decomposition. The traditional way of doing this was to use the Gaussian elimination, which requires $O(N^3)$ time to compute an LU decomposition. This obviously does not scale well.

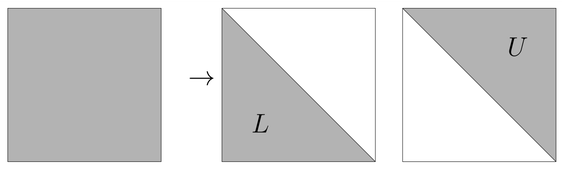
In order to reduce the time to computation, we use a hierarchical representation of the matrix using low rank and full rank blocks that can compute the LU decomposition in $O(N)$ time with some error due to approximation. The technique that we use allows tuning the accuracy of calculation as a trade off for speed.

Introduction

LU Decomposition

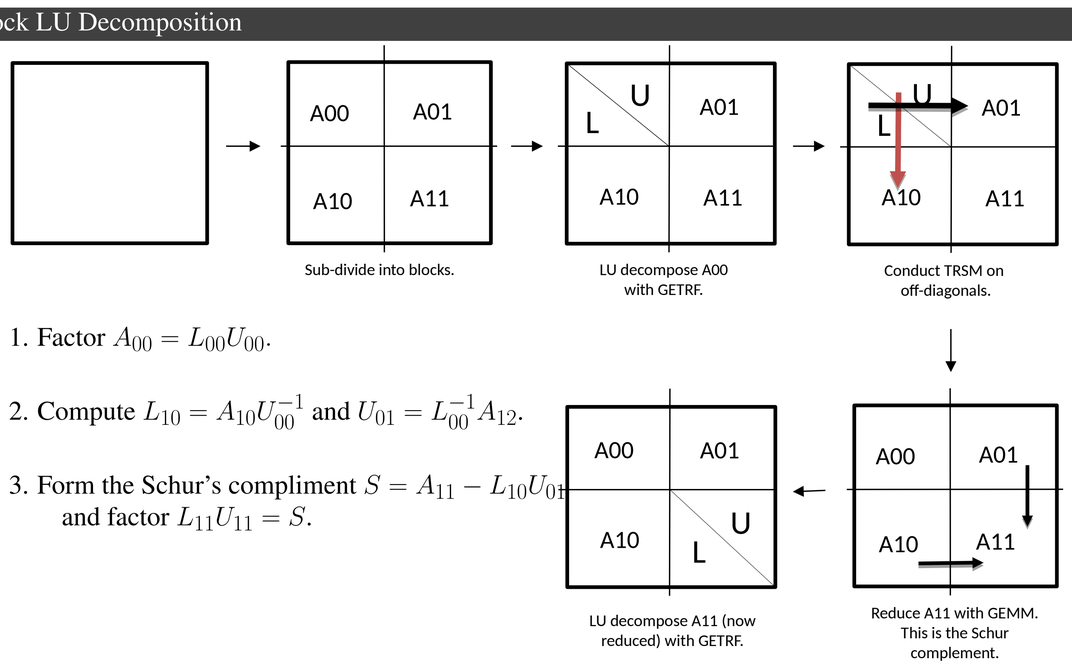
The LU decomposition is a very widely used matrix transformation in many operations ranging from Deep Learning to Computational Fluid Dynamics. It works by splitting a dense matrix into a lower and upper diagonal matrix.

Regular LU Decomposition



The regular method is usually used for smaller matrices where much optimization is not necessary.

Block LU Decomposition

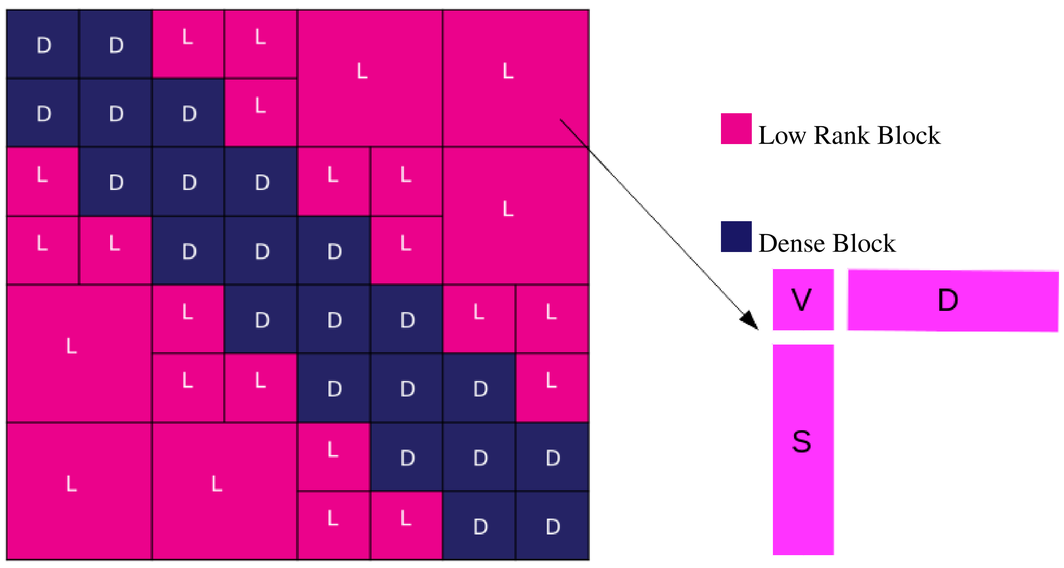


1. Factor $A_{00} = L_{00}U_{00}$.
2. Compute $L_{10} = A_{10}U_{00}^{-1}$ and $U_{01} = L_{00}^{-1}A_{12}$.
3. Form the Schur's compliment $S = A_{11} - L_{10}U_{01}$ and factor $L_{11}U_{11} = S$.

The Block LU Decomposition works by splitting the larger matrix into blocks, where each block can be computed individually. This more useful for larger matrices and is the basis for our technique.

The Gaussian Elimination method requires $O(N^3)$ time for computing the LU decomposition. The limitations of this approach are quickly realized if *N* is very large.

Hierarchical Matrix



The Hierarchical Matrix allows us to express a dense matrix as Hierarchical blocks of Full Rank and Dense matrices. The Low Rank Matrix is represented as a product of the Singular Value Decomposition (SVD) of the corresponding dense matrix.

As can be seen in the above figure, most of the information of the matrix is contained within the dense blocks (blue) that are centered around the diagonal of the matrix.

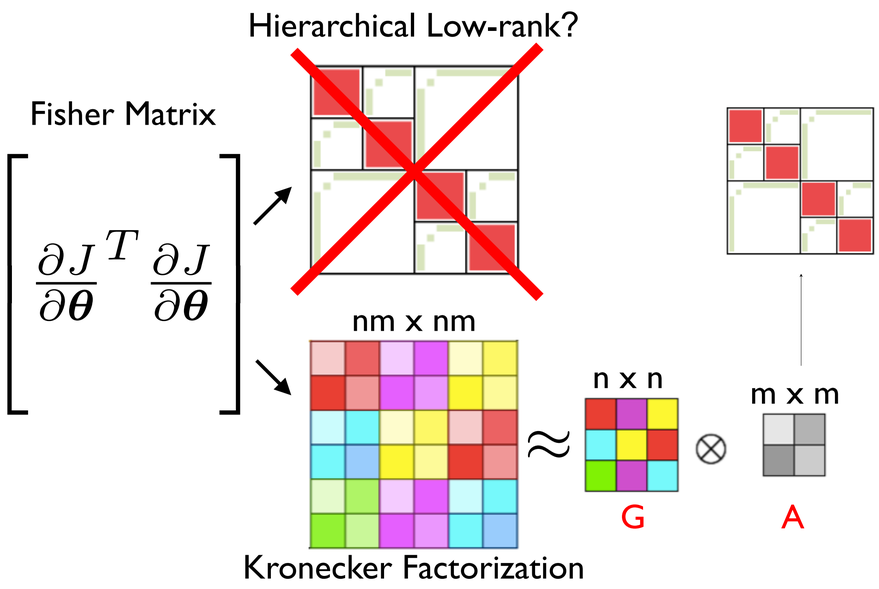
Use of Hierarchical matrices can reduce the cost of computation to $O(N)$ and the memory requirement to $O(N)$. This reduction in time and space can be achieved by maintaining an accuracy of 98%.

Kronecker Factorization

The Fisher Matrix is obtained from a second order optimization equation:

$$\theta_{\tau+1} = \theta_t - \eta \boxed{\left(\frac{\partial J^T}{\partial \theta}\frac{\partial J}{\partial \theta}\right)^{-1}} \frac{\partial J}{\partial \theta}$$

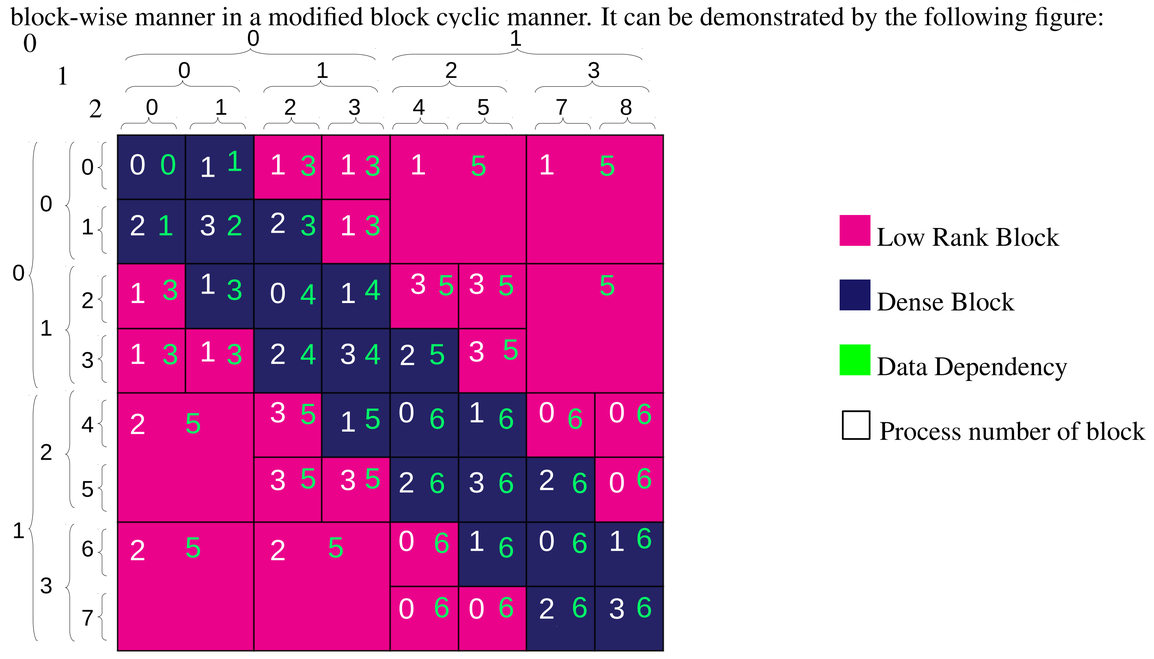
The boxed term is the Fisher matrix. It can have dimensions in the order of several millions. Inverting this matrix is a major challenge.



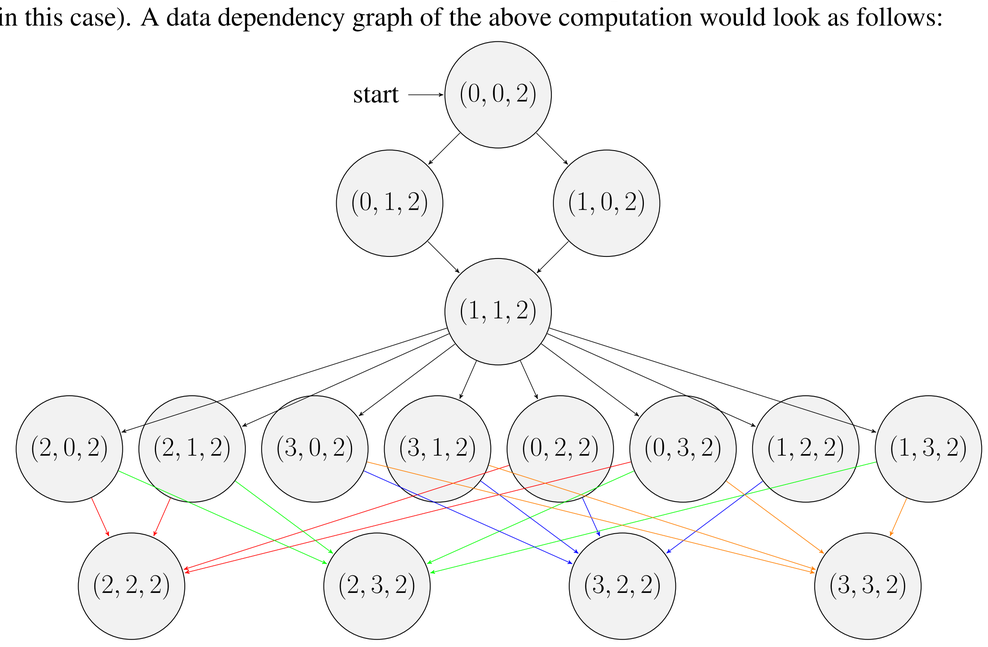
We plan to use the HMatrix LU decomposition technique on the A matrix above for the inversion.

Distributed Hierarchical Matrix

The LU decomposition has a hard data dependency on the top-left corner block. The computation cannot proceed unless the top left block has finished computation. The distribution of the matrices happens in a block-wise manner in a modified block cyclic manner. It can be demonstrated by the following figure:



As can be seen in the above figure, the green numbers indicate the data dependency and the white numbers indicate the process on which the particular block exists. The level of the hierarchical matrix is indicated by the numbers on the top left corner. The actual computation takes place only at the lowest level (level 2 in this case). A data dependency graph of the above computation would look as follows:



The Low Rank blocks exist on the same processor because they do not occupy a lot of data, which reduces the communication cost. The cost of performing a `gemm` (dense matrix multiplication) or `trsm` (dense matrix LU decomposition) is much lesser than cost of communication.

We still distribute each dense block over different processes because the space requirement of the dense block is large and time for computing `gemm` and `trsm` computations is non-trivial.

Future Work

In the future we will work on optimizing the library such that the `gemm` and `trsm` calls are optimum and communication is fully optimized for matrices of any dimension.

Conclusion

So far in our research, using Hierarchical matrices for solving large problems seems to be a promising new way of accelerating training of Deep Neural Networks.