Key Concepts For Parallel Out-Of-Core LU Factorization

David W. Walker
Mathematical Sciences Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6367

walker@msr.epm.ornl.gov

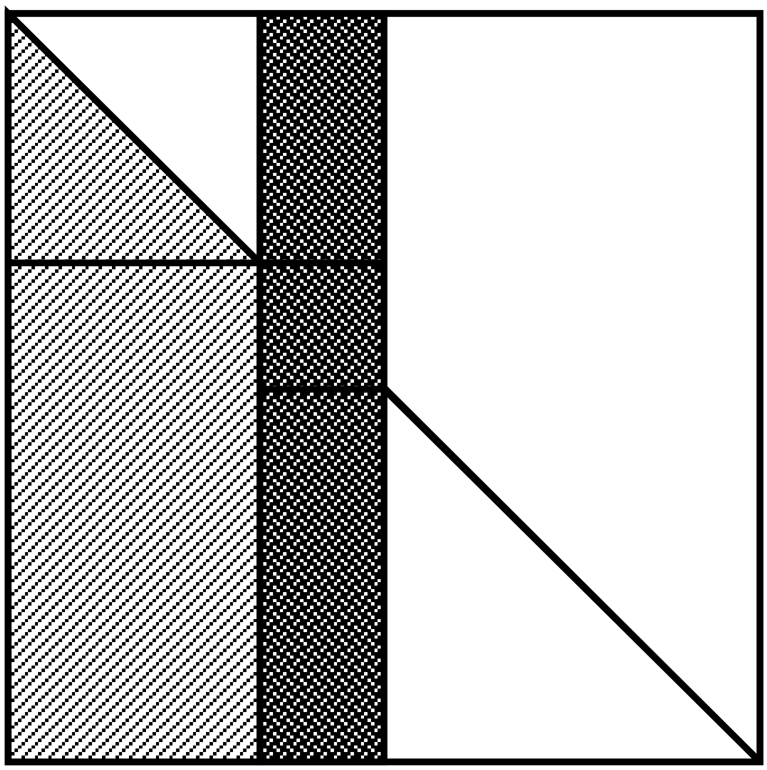
WEB: http://www.epm.ornl.gov/~walker/

Presented at
University of Tennessee, Knoxville, TN
October 20, 1995

Overview

- Left-looking and right-looking LU factorization algorithms
- Left-looking out-of-core algorithms
- Parallel I/O considerations
- A library of parallel I/O operations for linear algebra
- The hybrid parallel out-of-core algorithm
- Summary

Left-Looking LU Factorization

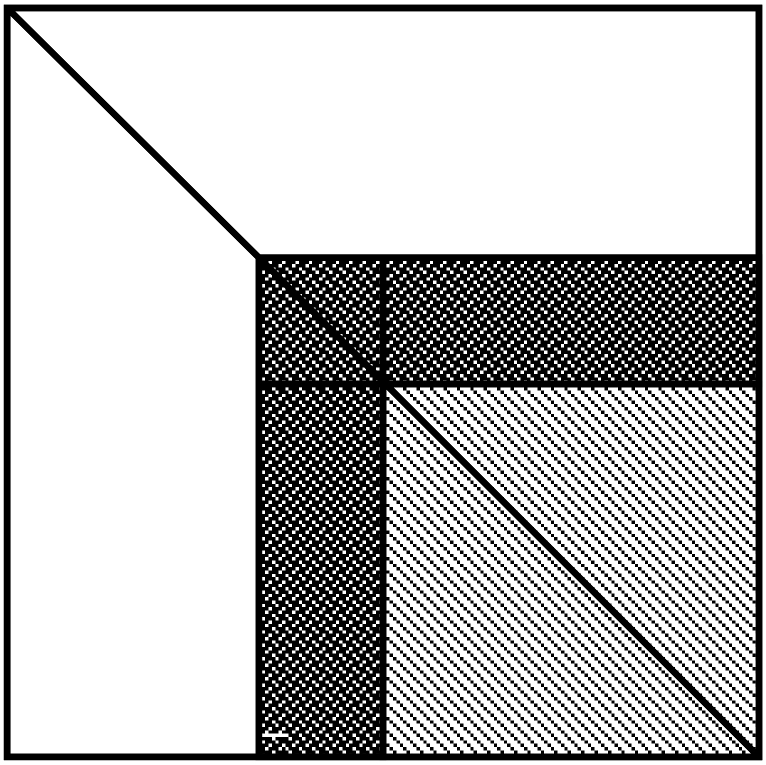


Left–looking variant

The main computational phases of the left-looking algorithm are:

1. Triangular solve
2. Matrix-matrix multiplication
3. Factorization

Right-Looking LU Factorization



Right–looking variant

The main computational phases of the right-looking algorithm are:

1. Factorization
2. Triangular solve
3. Matrix-matrix multiplication

I/O Cost For Left-Looking

Two variants of left-looking algorithm:

- Store matrix in unpivoted form
  - need to read complete column blocks
  - apply pivots "on the fly"
  - no need to re-write columns to left
- Store matrix in pivoted form
  - only need to read partial column blocks to left (on and below diagonal)
  - no need to apply pivots "on the fly"
  - must write back partial columns to left after applying pivots from previous factorization

Left-Looking Reads and Writes

For the version storing matrix in unpivoted form the I/O cost is

$$\frac{M^3}{2n_b}\left(1 + O(n_b/M)\right) R + 2M^2\left(1 + O(n_b/M)\right) W$$

so writes can be ignored.

For the version storing matrix in pivoted form the I/O cost is

$$\frac{M^3}{3n_b}\left(1 + O(n_b/M)\right) R + \frac{M^3}{6n_b}\left(1 + O(n_b/M)\right) W$$

- If reads and writes cost the same then the I/O cost of the two versions are equal.

Right-Looking Reads and Writes

The I/O cost is

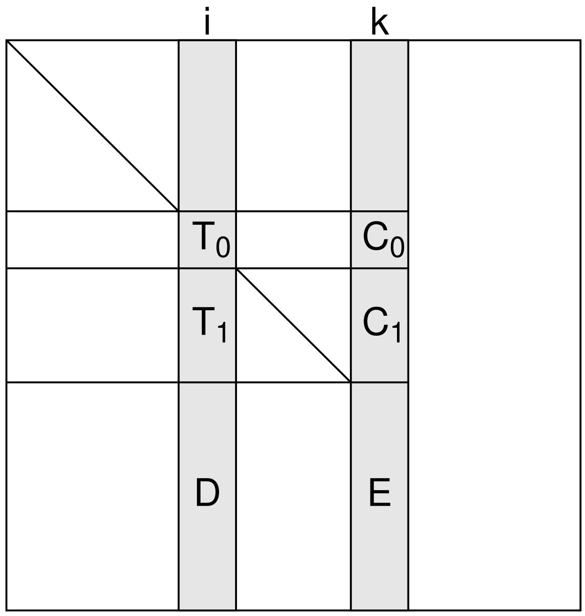
$$(R+W)\sum_{k=0}^{M/n_b-1}(M-kn_b)^2=\frac{M^3}{3n_b}\left(1+O(n_b/M)\right)(R+W)$$

Note that

- Left-looking algorithm has lower I/O cost
- Right-looking algorithm contains more parallelism

Out-Of-Core Algorithm

Assume we can have only two block columns in-core, and partition them thus:



What is the role of block column i in updating block column k?

Out-Of-Core Updates

Block column i plays the following role:

$$\begin{aligned} C_0 &\leftarrow T_0^{-1} C_0 \\ C_1 &\leftarrow C_1 - T_1 C_0 \\ E &\leftarrow E - D C_0 \end{aligned}$$

This can be expressed as:

$$\begin{pmatrix} C_1 \\ E \end{pmatrix} \leftarrow \begin{pmatrix} C_1 \\ E \end{pmatrix} - \begin{pmatrix} T_1 \\ D \end{pmatrix} C_0. \tag{1}$$

- The triangular solve and the matrix-matrix multiplication steps are mingled.

Left-Looking Pseudocode

```
go to start of file
for (each block column, k=0,1,...,M/n_b-1)
   read block column k into active block
   _LASWP : apply pivots to active block
   go to start of file
   for (each block column to left, i=0,1,...k-1)
      read block column i into temporary block
      _LASWP : apply pivots to temporary block
      _TRSM  : triangular solve
      _GEMM  : matrix multiply
   end for
   _GETRF : factor lower portion of active block
   _LASWP : unpivot active block
   write active block
end for
```

Figure 1: Pseudocode for out-of-core, left-looking LU factorization algorithm that leaves matrix in unpivoted form.

Approaches To Parallel I/O

Divide parallel I/O modes into two broad classes

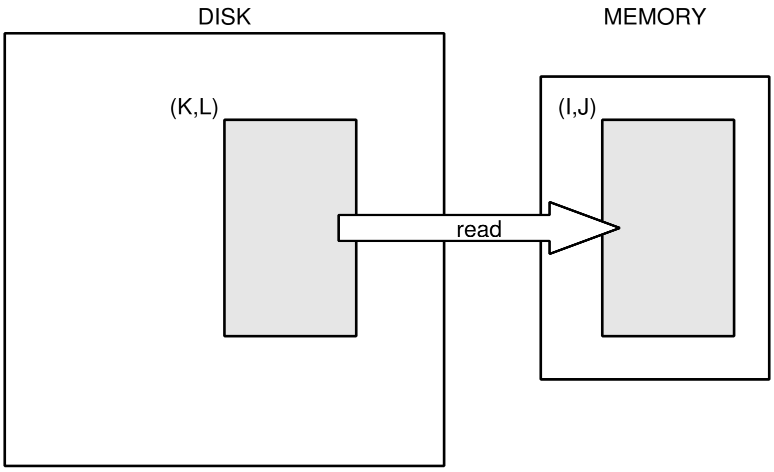
1. There is one file pointer into the disk file. In this case there are two possibilities
   (a) Only one process has access to the file pointer. Thus only that process can do I/O to the file, and has to scatter to, or gather from, the other processes when reading or writing the file.
   (b) All processes in a group have access to the file pointer. Synchronization is required if the order in which data are written to, or read from, the file is important.
2. Each process in a group has its own file pointer. Again there are two main possibilities
   (a) The file pointers can all access a global file space. In this case we refer to the file as a "shared file."
   (b) Each file pointer can only access its own local file space. In this case we refer to the file as a "distributed file."

Alignment

Suppose we want to read a submatrix

- from location (K, L) from an out-of-core matrix
- to location (I, J) in an in-core matrix

where locations are global indices.



In-core matrix is described by a descriptor

What should the file pointer of each process be set to?

This is determined by aligning (K, L) with (I, J).

BLAPIO Library

The Basic Linear Algebra Parallel I/O library contains the following routines for opening and closing files and manipulating file pointers.

POPEN. Open a file.

PCLOSE. Close a file.

P_ASEEK. Positions file pointers according to a specific alignment.

P_GSEEK. Positions file pointers according to an implicit alignment — assume out-of-core and in-core matrices have same data distribution, then (I, J) and (K, L) correspond to the same local indices within a block in the same process.

P_LSEEK. Positions file pointer to specific location.

BLAPIO Library

The Basic Linear Algebra Parallel I/O library contains the following routines for reading and writing.

P_READ. Reads submatrix into specified location of a matrix, and leaves the file pointer for each process at the next data element for the process. This is a blocking call.

P_WRITE. Writes submatrix from specified location of a matrix, and leaves the file pointer for each process at the next data element for the process. This is a blocking call.

P_IREAD. Reads submatrix into specified location of a matrix, and leaves the file pointer for each process at the next data element for the process. This is a nonblocking call.

P_IWRITE. Writes submatrix from specified location of a matrix, and leaves the file pointer for each process at the next data element for the process. This is a nonblocking call.

PIOTEST. Test if nonblocking parallel I/O call has completed.

PIOWAIT. Block until nonblocking parallel I/O call has completed.

BLAPIO Library

The Basic Linear Algebra Parallel I/O library contains the following routines for converting between shared and distributed files.

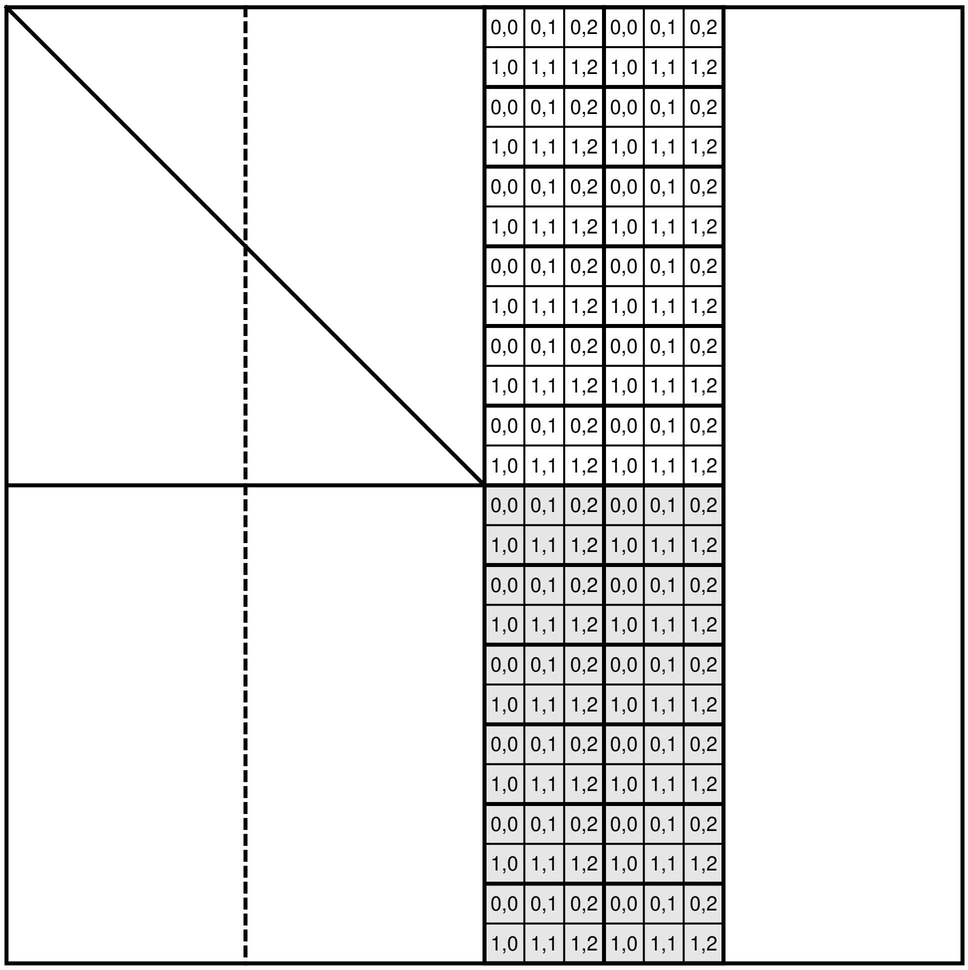
P_STOD. Converts a shared file to a distributed file.

P_DTOS. Converts a distributed file toa shared file.

Parallel Out-Of-Core Algorithm

- Left-looking algorithm has smaller I/O cost
- Right-looking has more parallelism
- The hybrid parallel out-of-core algorithm is left-looking at the highest level, but is right-looking within a block column
- In the hybrid algorithm the width of a block column is an integer number of process grids, n_g.
- In the hybrid algorithm all processes are involved in the triangular solve, the matrix-matrix multiplication, and the column factorization
- But, overall it's a left-looking algorithm

Structure Of Hybrid Algorithm



Parallel Left-Looking Pseudocode

Paralleization is simple and uses BLACS, PBLAS, ScaLAPACK, and BLAPIO as building blocks.

```
P_GSEEK : go to start of file
for (each superblock column, k=0,1,...,M/n_b-1)
   P_READ  : read superblock column k into active superblock
   P_LAPIV : apply pivots to active superblock
   P_GSEEK  : go to start of file
   for (each superblock column to left, i=0,1,...k-1)
      P_READ  : read superblock column i into temporary superblock
      P_LAPIV : apply pivots to temporary superblock
      P_TRSM  : triangular solve
      P_GEMM  : matrix multiply
   end for
   P_GETRF : factor lower portion of active superblock
   P_LAPIV : unpivot active superblock
   P_WRITE : write active superblock
end for
```

Figure 2: Pseudocode for parallel, out-of-core, left-looking LU factorization algorithm that leaves matrix in unpivoted form.

Summary and Future Work

- Two sorts of out-of-core file structures have been proposed — shared and distributed.
- Have written routines for converting between shared and distributed files.
- The BLAPIO library of parallel I/O operations for linear algebra has been designed and partially implemented
- A hybrid parallel out-of-core LU factorization routine has been written and tested on the Intel Paragon
- Need to complete implementation of BLAPIO (nonblocking routines)
- Need to do performance studies and test usefulness of nonblocking operations.
- Need to port BLAPIO to other platforms.