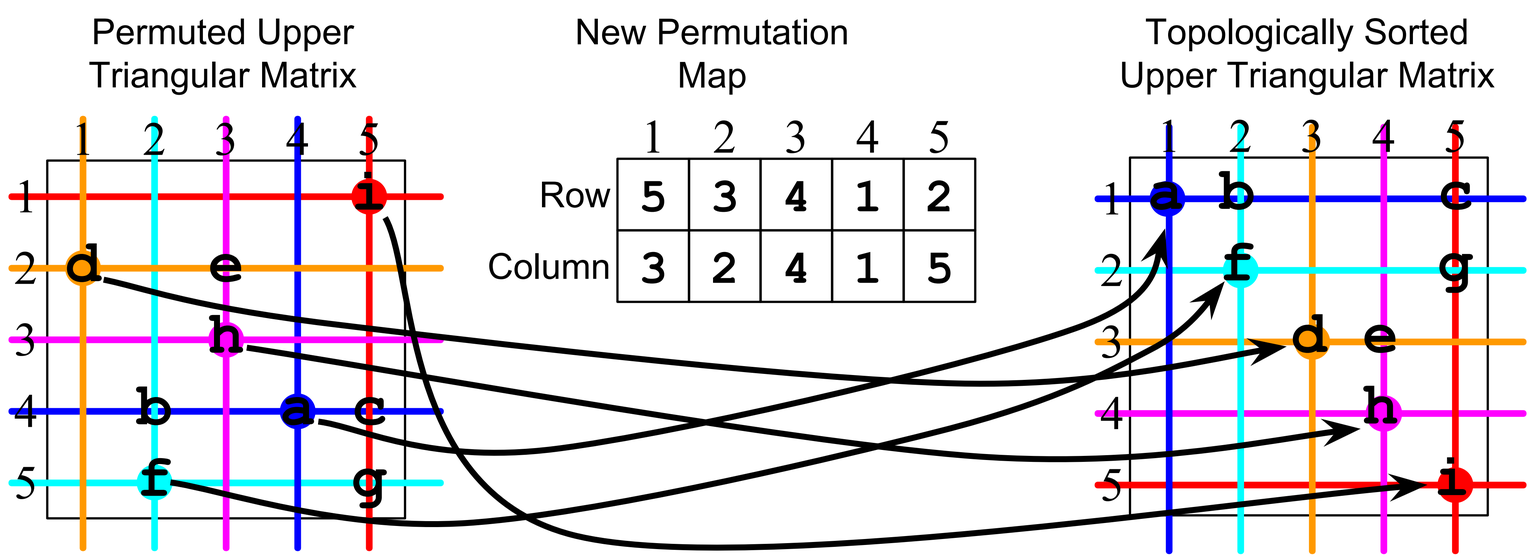
Sparse Matrix Topological Sorting in Chapel

Ian J. Bertolacci
University of Arizona
ianbertolacci@email.arizona.edu

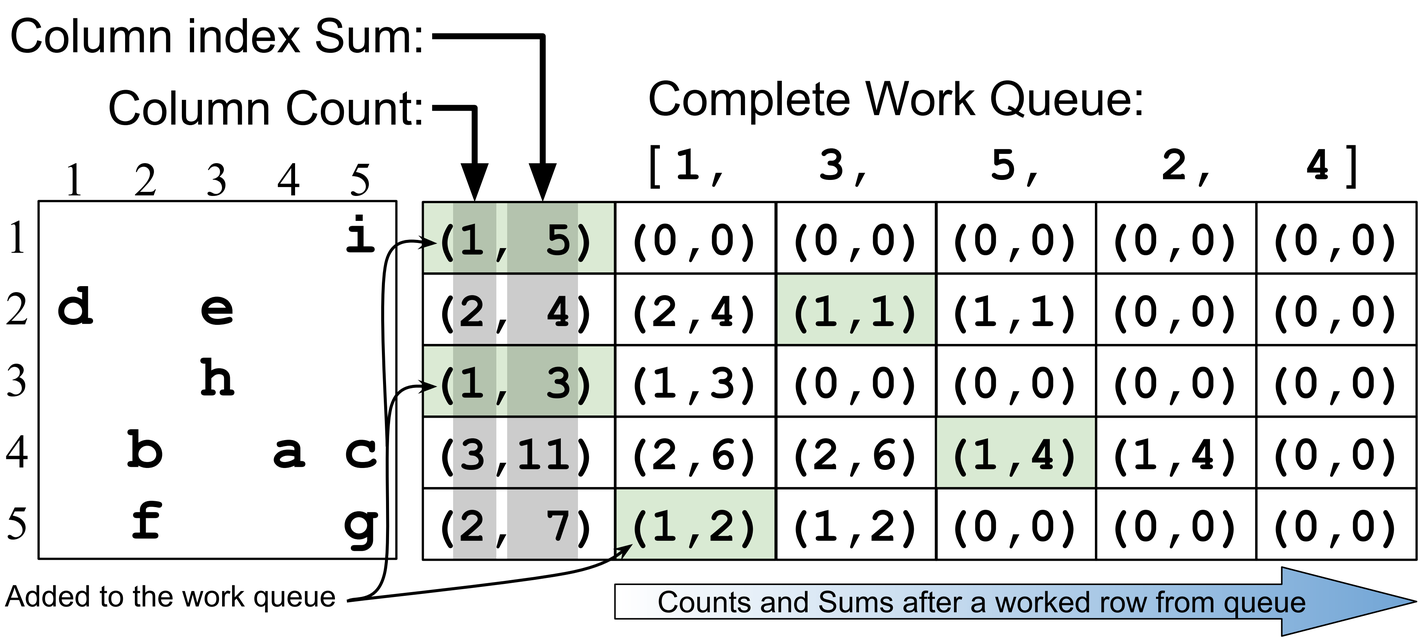
Goal: Convert a potential user's Sparse Matrix Toposort benchmark from UPC to Chapel to illustrate it's parallel functionality and sparse data-structures.

Matrix Toposort: Given permuted upper triangular square matrix, find *new* permutation map which converts it into a (possibly different) upper triangular matrix. Operates solely based on the domain (index set) of the matrix, specifically the non-zero indexes.

Permutation Map: Simply maps individual rows and columns to new rows and columns. All values in a row/column stay are mapped to the same new row/column.



Basic Algorithm: To initialize, for each row count the number of non-zero columns, and sum their index index values. In the core of the algorithm, for each row from the queue, the column index sum of that row is an 'unsorted' column. That row and column are mapped to the first available diagonal position, starting at (N,N) and ending with (1,1). The counts of all rows containing that column are decremented, and the column subtracted from the sum. When any row's count is reduced to 1, it is enqueued.



Current Performance Challenges: The sparse data structure must be optimized to allow contiguous iteration over columns (CSC works). Ordering of the work queue is highly dependent on the matrix, and deciding an ordering is equivalent to solving the algorithm. While all work in the queue can be performed in parallel (while protecting shared resources), the overhead relative to the work amount of work can dwarf the performance. In high density cases, thread scheduling is serialized as the queue is filled with few values (sometimes as few as 0 or 1), and in low density cases, it becomes embarrassingly parallel with almost no work per-thread. In both cases, a serial implementation can outperform a parallel implementation, often significantly.