For this assignment implement L-U Factorization using threads that make optimal use of an arbitrary number of processors that share the same memory. This assignment is intended to introduce you to or re-familiarize you with threaded programming and to illustrate the potential difficulties of doing performance-based design for programmable multiprocessor systems.

**L-U Factorization**

L-U Factorization is used to factor matrices in lieu of matrix inversion for solving linear simultaneous equations. This algorithm is an interesting mix of things that can happen in parallel, things that must happen in sequence, and unbalanced numbers of each. The computation in LU Factorization is simple, regular, and easily described. Given a system of linear simultaneous equations, \( Ax = b \), where \( A \) is an \( n \times n \) matrix and \( x \) and \( b \) are \( n \times 1 \) vectors, the system can be solved as \( x = A^{-1}b \). The tricky thing is inverting a dense \( A \) matrix. LU Factorization factors the \( A \) matrix as, 

\[
A = \begin{bmatrix}
    a_{11} & a_{12} & a_{13} & a_{14} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & a_{34} \\
    a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
= \begin{bmatrix}
    s_{11} & 0 & 0 & 0 \\
    s_{21} & s_{22} & 0 & 0 \\
    s_{31} & s_{32} & s_{33} & 0 \\
    s_{41} & s_{42} & s_{43} & s_{44}
\end{bmatrix} \times \begin{bmatrix}
    1 & t_{12} & t_{13} & t_{14} \\
    0 & 1 & t_{23} & t_{24} \\
    0 & 0 & 1 & t_{34} \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

The entries of the \( s \) and \( t \) matrices are given as

\[
t_{ij} = \frac{1}{s_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} s_{ik} \times t_{kj} \right)
\]

\[
s_{ij} = a_{ij} - \sum_{k=1}^{j-1} s_{ik} \times t_{kj}
\]

Once the \( A \) matrix is factored in this way, it is easy to use back substitution to solve for \( x \). The difficult thing is coming up with the \( s \) and \( t \) matrices.

Since the form of the \( s \) and \( t \) matrices is regular, the LU Factorization can be thought of as transforming the \( A \) matrix to a factored, compact form of the \( A \) matrix, \( F \), as

\[
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} & a_{14} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & a_{34} \\
    a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
    s_{11} & t_{12} & t_{13} & t_{14} \\
    s_{21} & s_{22} & t_{23} & t_{24} \\
    s_{31} & s_{32} & s_{33} & t_{34} \\
    s_{41} & s_{42} & s_{43} & s_{44}
\end{bmatrix}
\]
Some of the A→F transformation can be done in parallel and some requires sequencing of the calculations. The ordering of calculations for a 4*4 is

\[
\begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 2 & 3 & 3 \\
0 & 2 & 4 & 5 \\
0 & 2 & 4 & 6
\end{bmatrix}
\]

Further, each computation step above requires only single pair of operations, either a subtract-multiply (-,*) or a subtract-divide (-,/) (it will be useful to prove this to yourself).

**Problem Description**

Design an implementation for the pipelined processing of *successive sets* of A→F LU-Factorizations. The pipelining of successive sets requires that sets of *different* $A_i$ matrix inputs are being processed simultaneously, as inputs are made available. The arrival of the $A$ matrix triggers processing as it would in a hardware-like dataflow model as

\[
A_i \xrightarrow{\text{processing}} \ldots \xrightarrow{\text{processing}} F_i
\]

If we were doing this in Verilog, the entire $A$ matrix might be on the sensitivity list of a giant non-blocking always block that does the transformation. There is flow from the inputs to the outputs but also flow inside the algorithm. Only you do not have Verilog to do this — you’re not even sure how many processors you are going to have in your system!

Use your favorite thread package to implement 4×4 LU Factorization such that

- factorization of a series of input matrices occurs in a pipelined manner — in steady-state, there should be different threads working on different input matrices.
- each thread contributes no more than one (-,*) or (-,/) pair of operations to the algorithm, i.e. the micro states are limited to single sets of these operations — parallel operations are implemented as thread-level macro-states.
- only mutexes and semaphores can be used to synchronize the threads — no priorities.
- the description takes advantage of a the maximum amount of parallelism available for a shared memory architecture where the number of processing elements ranges from 1 to $n$ as shown on the right.
- the description is written to be independent of the size of the $A$ matrix, although you only need to demonstrate your results for a set of 4*4 $A$ matrices.
Hints

We suggest that you use either Java or C/C++ with pthreads, or you are going to have to spend a few days teaching us your language!

You do not need to execute your solution on a multiprocessor. But, that just makes the challenge of showing why you think you have an optimal solution more interesting.

Start early, especially if you are new to threaded programming and synchronization concepts!

Finally, here is a valid A $\rightarrow$ F transformation to get you started. By the way, not all matrices are invertible! Watch for divide by zero when this is the case.

$$\begin{bmatrix} 2 & 4 & 3 & 5 \\ 6 & 9 & 12 & 13 \\ 7 & 10 & 14 & 15 \\ 11 & 8 & 16 & 17 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 2 & 1.5 & 2.5 \\ 6 & -3 & -1 & 0.67 \\ 7 & -4 & -0.5 & -0.33 \\ 11 & -14 & -14.5 & -6 \end{bmatrix}$$

Turn in

- (25%) Well-commented source code.
- (25%) Sample execution runs — at least eight 4*4 L-U Factorizations in succession (you are responsible for coming up with the matrices you will factor).
- (25%) A short, written discussion of your solution, including a performance analysis that demonstrates why you think you have an optimal solution. Include charts or formulae to show why your solution is optimal.
- (25%) Powerpoint slides that graphically illustrate your solution and (unique) analysis that you will present in class on the due date of the assignment.

Each of these parts will be evaluated according to how well they meet the criteria, above. Note that the analysis is what will make your presentation interesting to the class!

You will be given instructions regarding how to hand in your code as the due date approaches.

Note

This assignment is an exercise in parallelization of an algorithm and analysis of solution techniques. The operations in L-U Factorization are considerably simpler that would likely be used in a real solution that used pthreads; the overhead due to parallelization exceeds the complexity of the operations. Nonetheless, the point of the assignment is to examine different forms of parallelization on a real calculation and to analyze, and so generalize, your solution.