Implementation

I implemented the following algorithm, from the Project specification:

For converting matrix to upper-triangular matrix:

```
Procedure G_E (A, b, y) begin
    for k := 1 to N do begin
        for j := k+1 to N do begin
            A[k,j] := A[k,j]/A[k,k]
        endfor
        y[k] := b[k]/A[k,k]
        A[k,k] := 1
        for i := k+1 to N do begin
            for j := k+1 to N do begin
                A[i,j] := A[i,j] - A[i,k]×A[k,j]
            endfor
            b[i] := b[i] - A[i,k]×y[k]
            A[i,k] := 0
        endfor
    endfor
end
```

For back substitution:

```
Procedure Back_Substitution (U, x, y) begin
    for k:= N down to 1 do begin
        x[k] := y[k]
        for i := k-1 down to 1 do begin
            y[i] := y[i] - x[k]×U[i,k]
        endfor
    endfor
end
```

Approach

I first implemented this sequentially, and examined which loops would be parallelizable. I decided to parallelize the two inner for loops (although the second one is nested, I did not parallelize it). In the loops I chose, each iteration is independent of the other iterations of the loop, so they could be executed in parallel.

I was hoping that the Python multiprocessing library would be a good fit for this, but the library comes with many constraints, and although in theory I had parallelized the loops correctly, the overheads of this library are high and ended up slowing my program down instead.
In order for multiprocessing to be beneficial, the parallel task needs to be significant enough to save more time than is created by the overhead that is part of multiprocessing. Hence, since it was already creating a lot of overhead, I decided to not parallelize the nested for loop inside the second inner for loop.

I plan on knowing a library better before I use it for the next project, or stick to the basic and straightforward C pthreads.