CS211 HW2

Due: 10/30/2023 23:59

Guideline:

1. The template code is on github: https://classroom.github.com/a/TGhiSVpP.
   You need to complete **mydtrsv** and **mydgetrf** and write a report. If you want to get bonus points, you can complete **mydgetrf_block**, **mydgemm** and modify **pad.txt**. Upload the report and your repository to github.
   On eLearn, you need to submit a link to your repository.
2. To compile the code, you can use **make**.
   Each time you start a new terminal, you need to run **export LD_LIBRARY_PATH=/act/opt/intel/composer_xe_2013.3.163/mkl/lib/intel64:$LD_LIBRARY_PATH** once.
   To test your code on header node, you can use **./main {function name} {n}**.
   To evaluate your code on compute nodes, you can use **srun main {function name} {n}**.
   Or, you can use **starter.py** to run your code. In that case, the additional **export** command is unnecessary.
3. Padding can be used to avoid corner cases when running **mydgetrf_block**. Input your padding value in **pad.txt**.
4. We use **main.c** to measure the correctness, running time and performance.
5. We use **hpc-001** to test your performance when grading.

Problem 1 (10 points):

Perform a non-blocked LU factorization $A = LU$ for the following matrix A without pivoting:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 13 & 18 \\ 7 & 54 & 78 \end{bmatrix}$$

Your calculation process should show the intermediate state of A after each step of elimination.

Hint: The non-zero elements of L and U are positive integers no larger than 10.

Problem 2 (80 points):

Attached is a Matlab program to solve the general linear system $Ax = B$and verify the solution with the Matlab build-in solver. There are two approaches in this problem to complete the same task. The verification is done in **main.c**.

(1) Call the function **LAPACKE_dgetrf** in LAPACK (http://www.netlib.org/lapack) to perform the LU factorization of the coefficient matrix A and then call the function **cblas_dtrsv** to perform the forward substitution first and then call it again to perform the backward substitution. This method has been implemented in the template code.

(2) Call **mydgetrf** and **mydtrsv** implemented by yourself to perform the LU factorization, forward substitution, and backward substitution.

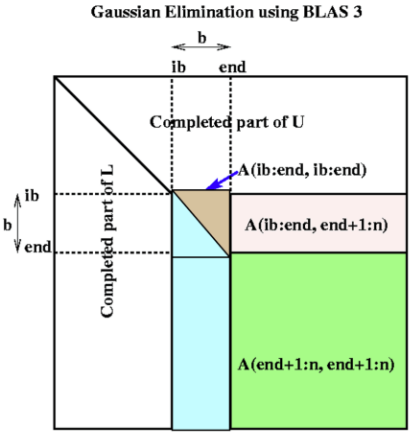
Your C functions **mydgetrf** and **mydtrsv** should follow the same algorithm as the Matlab code, but can have differences in implementation. Test your implementation and LAPACK version with matrix size 1000, 2000, 3000, 4000, 5000. Compare the performance (i.e., Gflops) of the two approaches.

```
function mylu(n)

A=randn(n,n); b=randn(n,1); Abk=A; pvt = 1:n;
%Factorize A. Your task: transform this part to mydgetrf().
for i = 1 : n-1,
   % pivoting %
   maxind=i; max=abs(A(i,i));
   for t=i+1:n,
      if ( abs(A(t,i))>max )
         maxind=t; max=abs(A(t,i));
      end
   end
   if (max==0)
     disp ( 'LUfactoration failed: coefficient matrix is singular' ); return;
   else
      if (maxind ~= i )
         %save pivoting infomation
         temps=pvt(i);
         pvt(i)=pvt(maxind);
         pvt(maxind)=temps;
         %swap rows
         tempv=A(i,:);
         A(i,:)=A(maxind,:);
         A(maxind,:)=tempv;
      end
   end
   %factorization
   for j = i+1 : n,
       A(j,i) = A(j,i)/A(i,i);
       for k = i+1 : n,
           A(j,k) = A(j,k) - A(j,i) * A(i,k);
       end
   end
end
%verify my factorization with Matlab for small matrix by printing results on screen. May skip in your code
myfactorization=A
mypivoting=pvt
[Matlab_L, Matlab_U, Matlab_P]=lu(Abk)

%forward substitution. Your task: transform this part to mydtrsm().
y(1) = b(pvt(1));
for i = 2 : n,
   y(i) = b(pvt(i)) - sum ( y(1:i-1) .* A(i, 1:i-1) );
end
%   back substitution. Your task: transform this part to mydtrsm().
x(n) = y(n) / A(n, n);
for i = n-1 : -1 : 1,
   x(i) = ( y(i) - sum ( x(i+1:n) .* A(i, i+1:n) ) ) / A(i,i);
end
%Matlab solve. Your task: call dgetrf() to factorize and dtrsm() twice (back and forward substit.) to solve.
xx= Abk\b;
%verify my solution with matlab. Your task: verify your solution with the solution from LAPACK.
Solution_Difference_from_Matlab=norm(x'-xx)
```

Problem 3 (10 points):
The following slide shows the blocked GEPP algorithm.

Blocked GEPP (www.netlib.org/lapack/single/sgetrf.f)

```
for   ib = 1 to n-1 step b     ... Process matrix b columns at a time
      end = ib + b-1           ... Point to end of block of b columns
      apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P' * L' * U'
      ... let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
      A(ib:end , end+1:n) = LL^-1 * A(ib:end , end+1:n)      ... update next b rows of U
      A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
                - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                              ... apply delayed updates with single matrix-multiply
                              ... with inner dimension b
```



Perform this algorithm on the following matrix A with block size $b = 2$ and without pivoting to achieve the LU factorization result $A = LU$:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 41 & 49 \\ 5 & 40 & 107 & 135 \end{bmatrix}$$

Your calculation process should show the intermediate state of A after each step (line) of the blocked GEPP algorithm.
Hint: The non-zero elements of L and U are positive integers no larger than 10.

Bonus Problem (20 points):
Implement the blocked GEPP algorithm **mydgetrf_block**.
Solve the linear system through your blocked version code using your own matrix multiplication code **mydgemm** and **mydtrsv** in problem 2.
Optimize your code to achieve as high performance as possible using any other techniques available to you (such as changing block size). Compare your performance with your un-optimized version in problem 2.