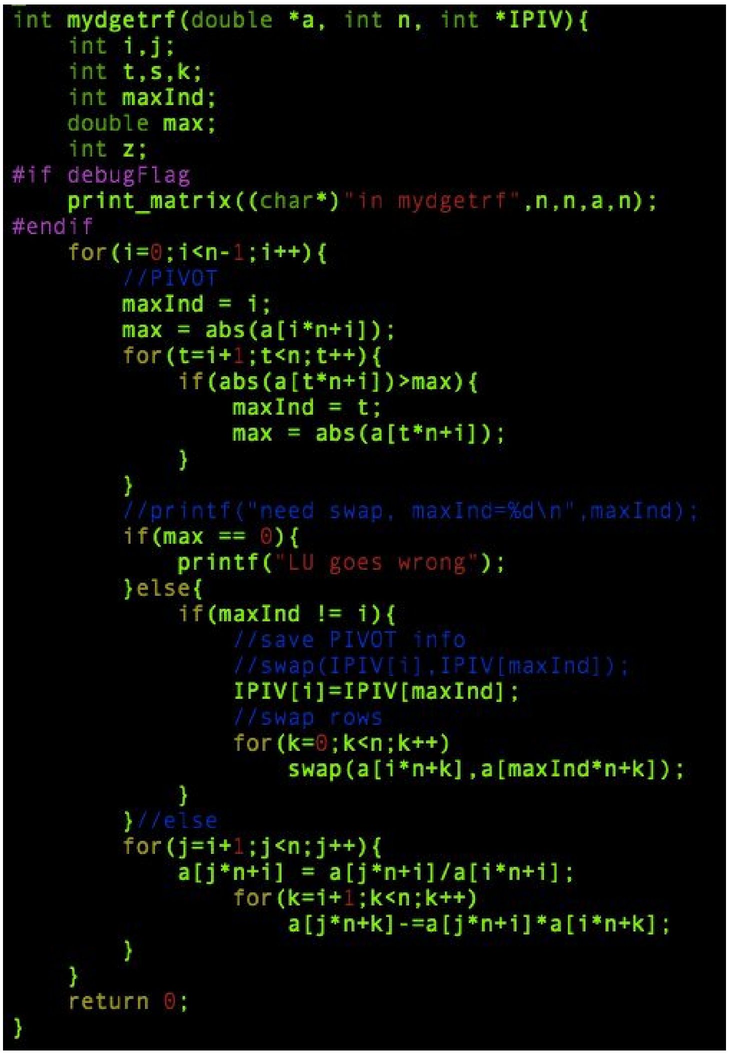
Name: Chen-yang Yu

Student ID: 862052273

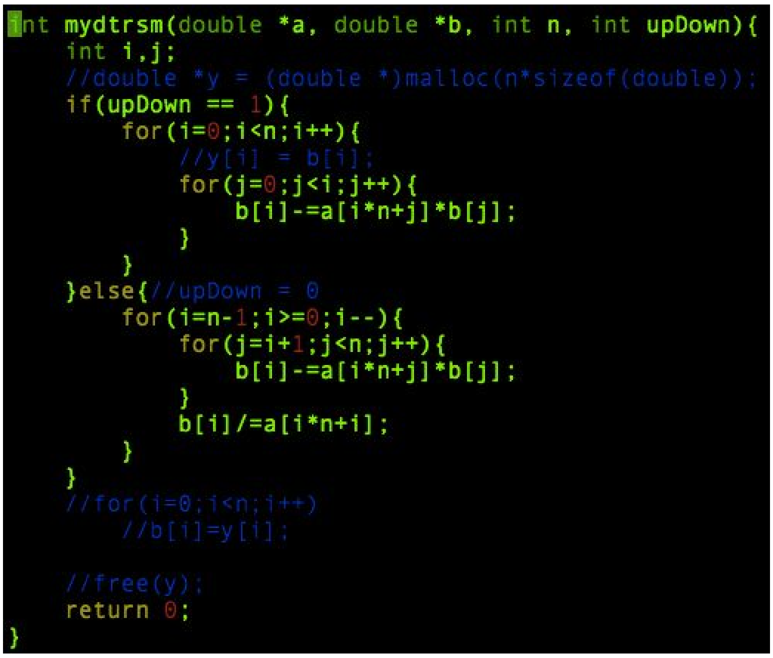
1.

(1) Approach_1: call function provided by LAPCKE

(2) Approach_2: call mydgetrf() to GEPP and call mydtrsm() twice to solve x. Check the correctness with the result of mydrtsm() and dtrsm_().



```
int mydgetrf(double *a, int n, int *IPIV){
    int i,j;
    int t,s,k;
    int maxInd;
    double max;
    int z;
#if debugFlag
    print_matrix((char*)"in mydgetrf",n,n,a,n);
#endif
    for(i=0;i<n-1;i++){
        //PIVOT
        maxInd = i;
        max = abs(a[i*n+i]);
        for(t=i+1;t<n;t++){
            if(abs(a[t*n+i])>max){
                maxInd = t;
                max = abs(a[t*n+i]);
            }
        }
        //printf("need swap, maxInd=%d\n",maxInd);
        if(max == 0){
            printf("LU goes wrong");
        }else{
            if(maxInd != i){
                //save PIVOT info
                //swap(IPIV[i],IPIV[maxInd]);
                IPIV[i]=IPIV[maxInd];
                //swap rows
                for(k=0;k<n;k++)
                    swap(a[i*n+k],a[maxInd*n+k]);
            }
        }//else
        for(j=i+1;j<n;j++){
            a[j*n+i] = a[j*n+i]/a[i*n+i];
                for(k=i+1;k<n;k++)
                    a[j*n+k]-=a[j*n+i]*a[i*n+k];
        }
    }
    return 0;
}
```



```
int mydtrsm(double *a, double *b, int n, int upDown){
    int i,j;
    //double *y = (double *)malloc(n*sizeof(double));
    if(upDown == 1){
        for(i=0;i<n;i++){
            //y[i] = b[i];
            for(j=0;j<i;j++){
                b[i]-=a[i*n+j]*b[j];
            }
        }
    }else{//upDown = 0
        for(i=n-1;i>=0;i--){
            for(j=i+1;j<n;j++){
                b[i]-=a[i*n+j]*b[j];
            }
            b[i]/=a[i*n+i];
        }
    }
    //for(i=0;i<n;i++)
        //b[i]=y[i];

    //free(y);
    return 0;
}
```

Gflops:

Since the algorithm is

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

When i = 1:

A(2:n,1) = A(2:n,1) / A(1,1) => (n-1) times operations

A(2:n, 2:n) = A(2:n, 2:n) - A(2:n,1) * A(1,2:n) => $(n-1)^2 \times 2$ times operations

When i = 2:

A(3:n,2) = A(3:n,2) / A(2,2) => (n-2) times operations

A(3:n, 3:n) = A(3:n, 3:n) - A(3:n,2) * A(2,3:n) => $(n-2)^2 \times 2$ times operations

…

When i = n-1:

A(n,n-1) = A(n,n-1) / A(n-1,n-1) => (n-2) times operations

A(n, n) = A(n, n) - A(n,n-1) * A(n-1,n) => $1^2 \times 2$ times operations

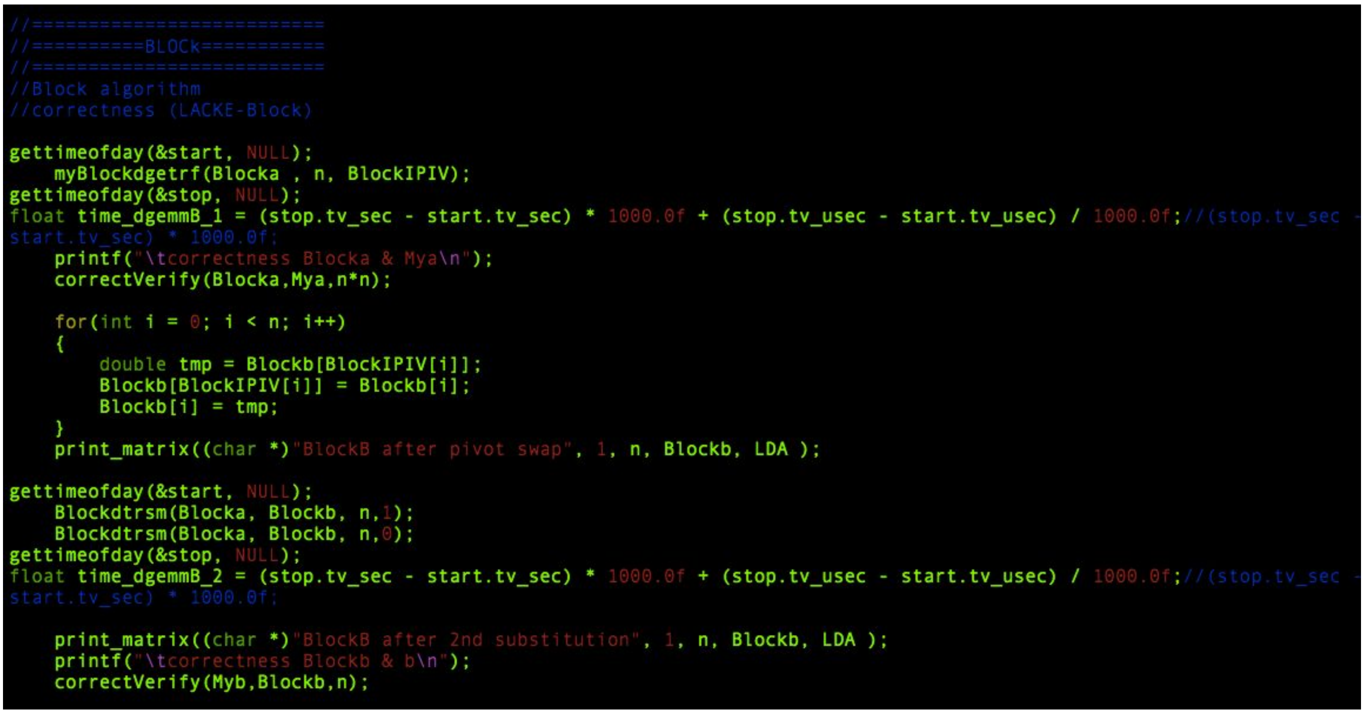
The total Gflops is

$$[1+2+3+\ldots+(n-1)]\times 1+[1^2+2^2+\ldots+(n-1)^2]\times 2=\frac{n(n-1)}{2}+\frac{(n-1)n(2n-1)}{3}=\frac{4n^3-3n^2-n}{6}$$

n	1000	2000	3000	4000	5000
LAPACK_dgetrf	487.154999	3993.951904	13828.932617	35279.023438	66935.468750
LAPCK_dtrsm	2.003000	9.099000	35.497002	64.402000	99.875000
mydgetrf	5003.225098	41274.972656	145320.28125	334212.96875	655849.56250
mydtrsm	9.679000	36.400002	86.602997	144.688995	241.574997
LAPACK_dgetrf	1.361993	1.333517	1.301293	1.207970	1.244295
mydgetrf	0.132615	0.129037	0.123833	0.127512	0.126992

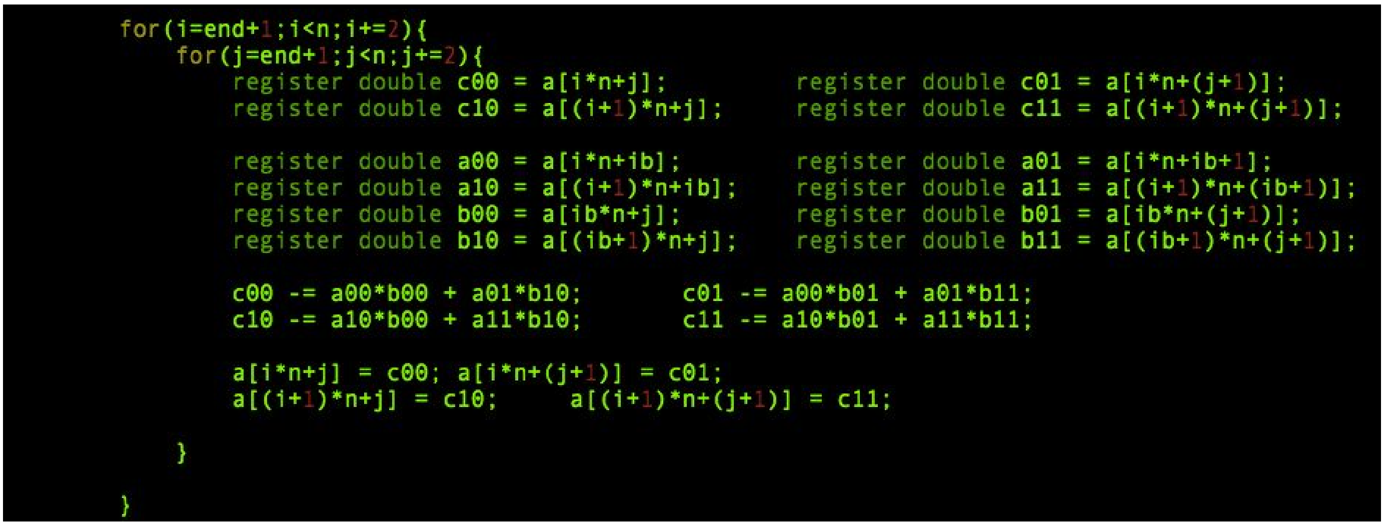
(millisec)
(Gflops)

2.
call myBlockdgetrf() to implement blocked GEPP and call Blockdtrsm() twice to solve x. Check the correctness with the result of mydgetrf() and myBlockdgetrf(). The performance comparison as table below.



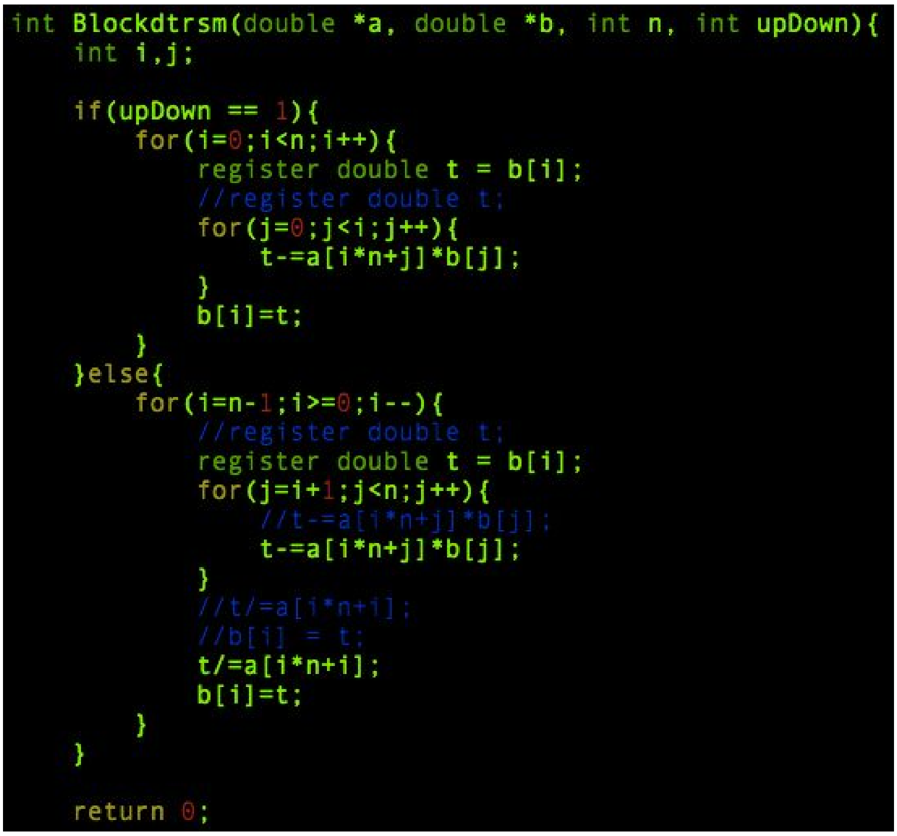
```
//===========================
//==========BLOCk===========
//===========================
//Block algorithm
//correctness (LACKE-Block)

gettimeofday(&start, NULL);
    myBlockdgetrf(Blocka , n, BlockIPIV);
gettimeofday(&stop, NULL);
float time_dgemmB_1 = (stop.tv_sec - start.tv_sec) * 1000.0f + (stop.tv_usec - start.tv_usec) / 1000.0f;//(stop.tv_sec -
start.tv_sec) * 1000.0f;
    printf("\tcorrectness Blocka & Mya\n");
    correctVerify(Blocka,Mya,n*n);

    for(int i = 0; i < n; i++)
    {
        double tmp = Blockb[BlockIPIV[i]];
        Blockb[BlockIPIV[i]] = Blockb[i];
        Blockb[i] = tmp;
    }
    print_matrix((char *)"BlockB after pivot swap", 1, n, Blockb, LDA );

gettimeofday(&start, NULL);
    Blockdtrsm(Blocka, Blockb, n,1);
    Blockdtrsm(Blocka, Blockb, n,0);
gettimeofday(&stop, NULL);
float time_dgemmB_2 = (stop.tv_sec - start.tv_sec) * 1000.0f + (stop.tv_usec - start.tv_usec) / 1000.0f;//(stop.tv_sec -
start.tv_sec) * 1000.0f;

    print_matrix((char *)"BlockB after 2nd substitution", 1, n, Blockb, LDA );
    printf("\tcorrectness Blockb & b\n");
    correctVerify(Myb,Blockb,n);
```

In myBlockdgetrf() (mydgetrf() block version), we use a block GEPP (block size = 2) and register reuse in the (n-2)*(n-2) matrix multiplication.



```
        for(i=end+1;i<n;i+=2){
            for(j=end+1;j<n;j+=2){
                register double c00 = a[i*n+j];         register double c01 = a[i*n+(j+1)];
                register double c10 = a[(i+1)*n+j];     register double c11 = a[(i+1)*n+(j+1)];

                register double a00 = a[i*n+ib];        register double a01 = a[i*n+ib+1];
                register double a10 = a[(i+1)*n+ib];    register double a11 = a[(i+1)*n+(ib+1)];
                register double b00 = a[ib*n+j];        register double b01 = a[ib*n+(j+1)];
                register double b10 = a[(ib+1)*n+j];    register double b11 = a[(ib+1)*n+(j+1)];

                c00 -= a00*b00 + a01*b10;       c01 -= a00*b01 + a01*b11;
                c10 -= a10*b00 + a11*b10;       c11 -= a10*b01 + a11*b11;

                a[i*n+j] = c00; a[i*n+(j+1)] = c01;
                a[(i+1)*n+j] = c10;     a[(i+1)*n+(j+1)] = c11;

            }

        }
```

In Blockdtrsm() (mydtrsm() block version), we use register reuse to speed up.



```
int Blockdtrsm(double *a, double *b, int n, int upDown){
    int i,j;

    if(upDown == 1){
        for(i=0;i<n;i++){
            register double t = b[i];
            //register double t;
            for(j=0;j<i;j++){
                t-=a[i*n+j]*b[j];
            }
            b[i]=t;
        }
    }else{
        for(i=n-1;i>=0;i--){
            //register double t;
            register double t = b[i];
            for(j=i+1;j<n;j++){
                //t-=a[i*n+j]*b[j];
                t-=a[i*n+j]*b[j];
            }
            //t/=a[i*n+i];
            //b[i] = t;
            t/=a[i*n+i];
            b[i]=t;
        }
    }

    return 0;
```

Gflops:

Since the algorithm is

```
For ib = 1 to n-1; ib+=2
    for i = ib to end
        A(i+1:n, i) = A(i+1:n, i) / A(i, i)
        A(i+1:n, i+1:end) -= A(i+1:n , i) * A(i , i+1:n)
    A(ib:end, end+1:n) = LL \ A(ib:end, end+1:n)
    A(end+1:n, end+1:n) = A(end+1:n, ibLend)*A(ib:end, end+1:n)
When ib = 1, end = 2
    When i = 1:
        A(2:n, 1) = A(2:n, 1) / A(1, 1) => (n-1) times operations
        A(2:n, 2) -= A(2:n, 1) * A(1, 2:) => (n-1)*2 times operations
    When i = 2:
        A(3:n, 2) = A(3:n, 2) / A(2, 2) => (n-2) times operations
    A(1:2, 3:n) = LL \ A(1:2, 3:n) => (n-2)*2 times operations
```

A(3:n, 3:n) = A(3:n, 1:2) * A(1:2, 3:n) => 2*2*(n-2)^2 times operations

Which is equal with the first two iteration of non-blocking algorithm. As the result, the totally number of double floating point operations is the same as $\frac{4n^3-3n^2-n}{6}$

Performance comparison between LAPCKE, mydgetrf, and blocked version (myBlockdtrsm)

n	1000	2000	3000	4000	5000
LAPACK_dgetrf	407.393982	3889.895020	11512.747070	29391.812500	61441.800781
LAPCK_dtrsm	2.023000	17.131001	29.497999	50.285000	55.939999
mydgetrf	578.093994	6720.421875	25453.466797	61949.390625	111835.43750
mydtrsm	6.569000	30.225000	65.705002	113.455002	184.968994
myBlockdgetrf	289.468994	2759.319092	10825.346680	26035.644531	57111.769531
myBlockdtrsm	2.185000	8.574000	26.313000	59.979000	75.699997
LAPACK_dgetrf	1.628649	1.369189	1.563093	1.449928	1.355551
mydgetrf	1.147740	0.792510	0.706996	0.687916	0.744733
myBlockdgetrf	2.292134	1.930187	1.662349	1.636833	1.458325

(millisec)
(Gflops)