Parallelizing the Method of Moments Code for Perfect Electric Conducting Cylinder for a Transverse Magnetic Incidence

Abdulla Aldilaijan

under the direction of

Prof. Hakan Bagci

PhD Student: Ali Imran

Computational Electromagnetics Laboratory

King Abdulla University of Science and Technology

July 9, 2014

**1 Purpose**

Computational simulations aim to demonstrate results of practical experiments before the experiments are executed. In this project, a code for predicting the scattering electric field off of a perfect electric conducting cylinder is translated from its original code in MATLAB to a serial code in Fortran. The serial Fortran code is then parallelized using Open MPI to be able to compute results using bigger amounts of data in less time. Through each step, results from the new code are compared to the older ones. The norm error is calculated, and the graphs are plotted. We also compare some of our results to the accurate mathematical calculations of the Mie series.

**1 Introduction**

Mathematics has provided us with a way to calculate and explain many physical problems. Arithmetic, geometry and algebra helped solve complex problems. Calculus explains some of the governing principles of this world. Mathematics keeps evolving to satisfy our needs.

Electromagnetics is governed by the Maxwell equations. Because of the linearity of these equations, problems in electromagnetics can be solved using analytical tools and theoretical modeling. Many Problems have been solved using with theoretical modeling include the Mie Series solutions.

However, due to the increase need to solve even further complex problems regarding complex objects, and because of the development of the computer, Computational Electromagnetics (CEM) [1] became an essential tool in solving electromagnetics equations more accurately. Furthermore, CEM helps us avoid expensive experimentations that mostly depend on trial and error. Because of its close representation and accuracy of the real world, CEM can be used to create virtual experiments.

CEM is made more efficient by using parallel computing. Two factors need to be considered in complex computation: cost and efficiency, and parallel computing provides the optimal compromise between the two [2]. Solving simple problems require a single computer. Solving complex problems would require more powerful single computers. Instead, exploiting the idea of parallel computing give us the ability to use several computers to solve a single problem efficiently with avoiding extra costs. This made parallel computing an obvious method, and has given rise to workstation networks to be used as a parallel computer. Even personal computers PC have started adopting this method of computing, which led to its increased performance and decreased prices.

**2 Problem Statement**

In this project, a MATLAB [4] serial source code is provided by the research group and is translated into a serial Fortran code, which is then parallelized. The code describes electromagnetic waves scattering from 2D impenetrable objects (Infinite cylinder in this case). The scattering problem is formulated using Electric Field Integral Equations (EFIE) incorporating transverse magnetic (TM) incidence. To find the scattered field and Radar Cross Section (RCS) of the fields, we find the current by solving a linear system of equations. This can be achieved by using the LU method, and the Iterative solver (TFQMR) method. Results from each method are compared to the Mie Series results for scattered field. The code is then parallelized using Open MPI. The main goal is to parallelize the operations of filling of the matrices, and the matrix multiplication. Results from the parallel program are compared to the results from the serial program.

**3 Process and Results**

**3.1 Serial Code**

The MATLAB code is translated into Fortran. Unlike MATLAB, which contains many intrinsic functions and scientific methods for solving problems, Fortran is limited. The program needs to incorporate external source codes to use Hankel function of second degree [5] and to find the solution to a system of equations. Two methods are used to find the solutions of a system of equations: a mathematical method, and an estimation method. Each method will yield different results.

**3.1.1 LU Method**

The LU method solves the system of equations mathematically. The zgesv subroutine from the Lapack library [3] is used to find the solution of a complex system of linear equations Ax=B using the LU method to find current. The results from the LU method is compared to the results from the original MATLAB program, and to another MATLAB program that computes the Mie Series solution, which is provided by the research group.

* Comparing the current of the LU method to the current of the original code yields the error norm of **4.4618e-6**

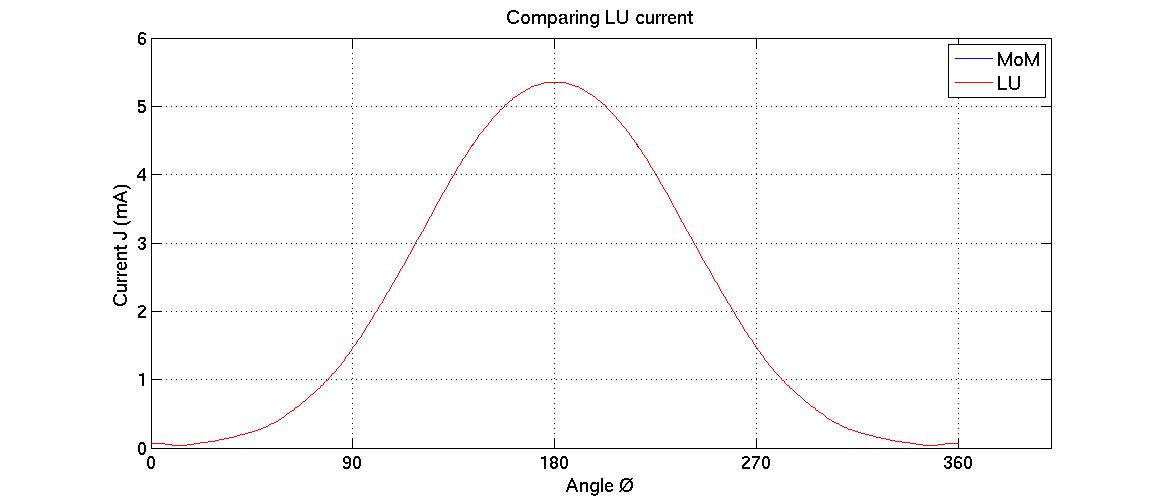


Figure Plot of current from original code, and LU method

* Comparing the scattered field from the LU method to the scattered field from the Mie Series code yields the error norm of **2.0e-3**, and comparing it to the scattered field of the original code yields the error norm of **3.2115e-8**

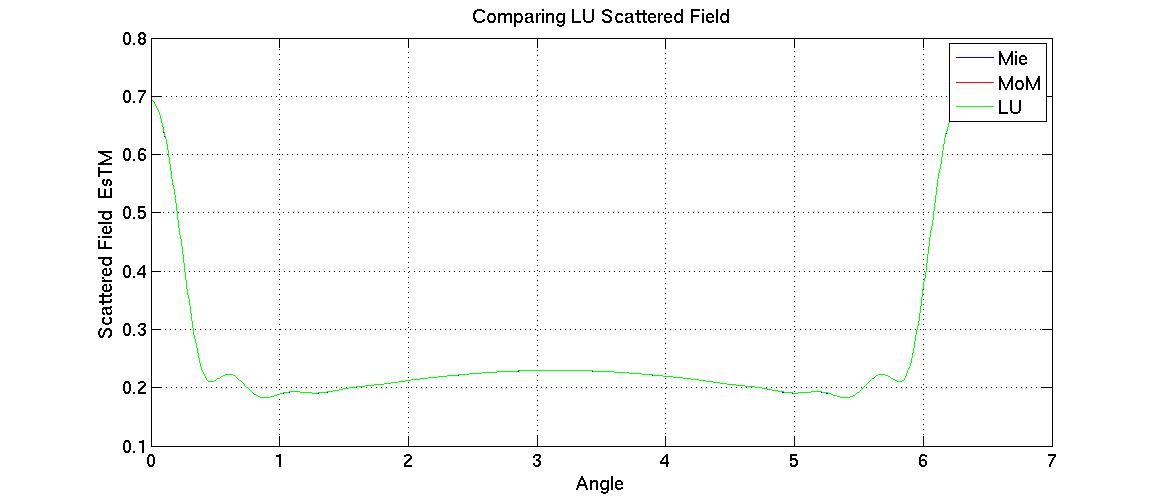


Figure Plot of scattered field from Mie Series code, original code, and LU method

**3.1.2 Iterative Method**

Although the LU method yields significantly accurate results, using the Iterative (TFQMR) method -provided by the research group- in the program allows for the parallelization of the code and enables the program to deal with huge amounts of data. The Iterative method solves the system of equations by repeating its calculations until either the error tolerance or the maximum number of iterations is reached. Results from the TFQMR method are first compared to the LU method based on error tolerance. Then, the result with the lowest error tolerance is used in comparison with the LU method, the original code, and the Mie Series code.

* Comparing different results of current from the TFQMR method based on error tolerance to current of the LU method yields the following plot

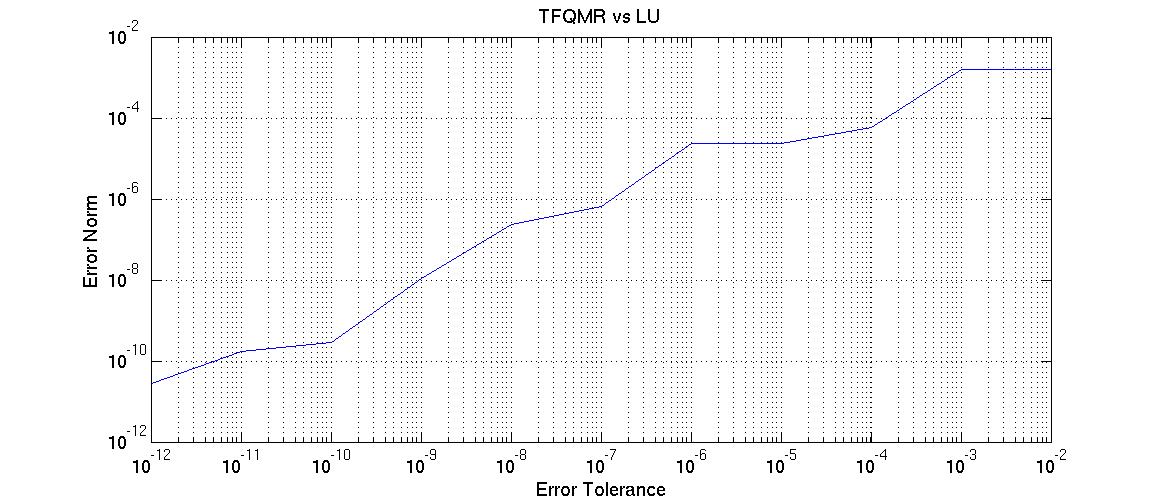


Figure Different results from TFQMR method based on error tolerance

* Comparing current of the TFQMR method (with 1e-12 tolerance) to current of the LU method yields the error norm of **2.5285e-11**

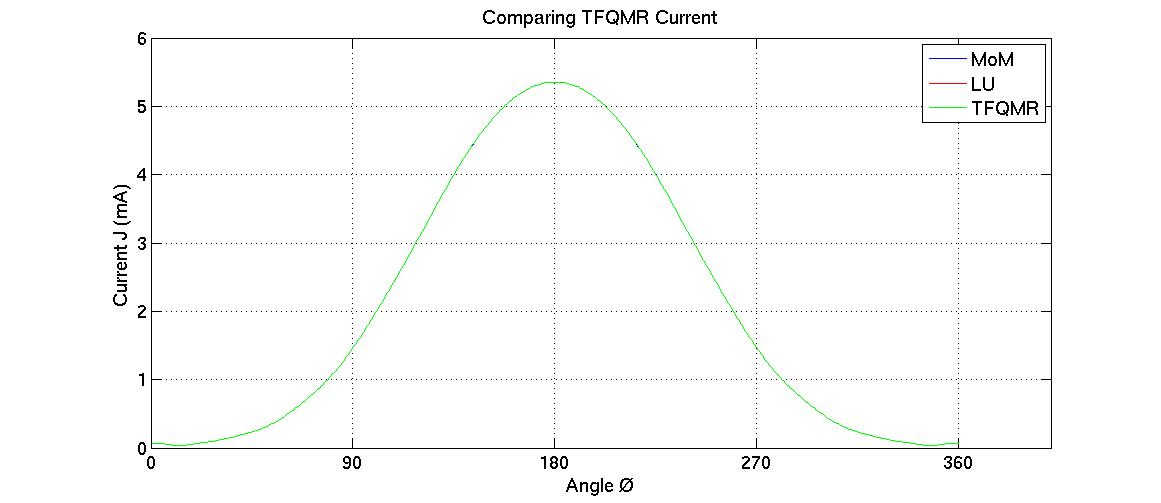


Figure Plot of current from original code, LU method, and TFQMR method

* Comparing the scattered field of the TFQMR method (with 1e-12 tolerance) to the scattered field of the Mie Series yields the norm error of **2.0e-3**, and to the scattered field of the LU method yields the error norm of **4.1168e-13**

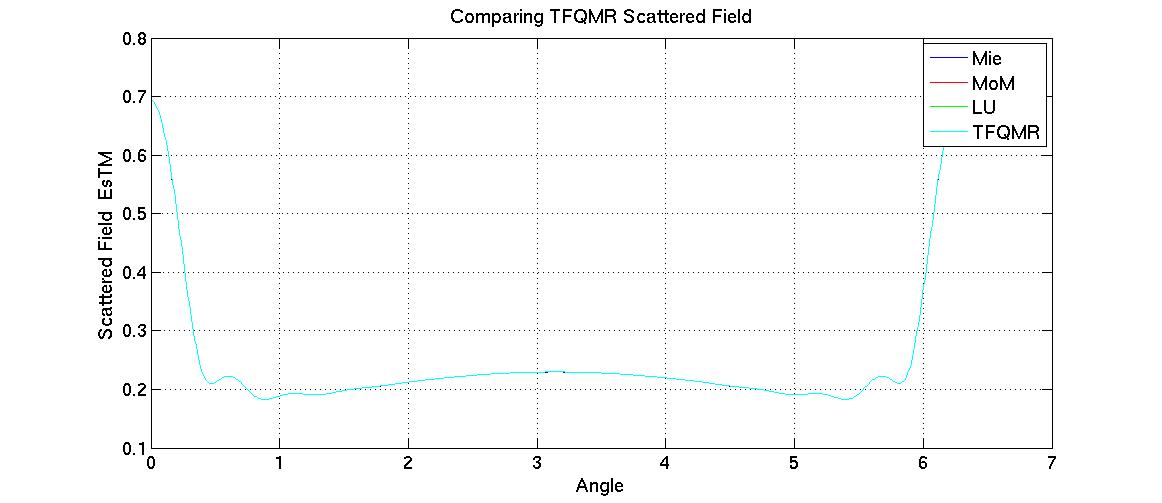


Figure Plot of scattered field from Mie Series, original code, LU method, and TFQMR method

**3.2 Parallel Code**

The code is parallelized using Open MPI. The parallelization occurs with filling the matrices needed to find the solution for the system of equations. Instead of having one matrix to hold all the entries, each process takes a chunk of the data, and stores it in a smaller matrix. The calculations in the iterative method are also parallelized. Each process takes part in the matrix multiplication. Both the Serial and Parallel results are based on the TFQMR method with error tolerance of 1e-12. The runtime of the code is observed by comparing the time for using different number of processes, in addition to the runtime of the serial code.

* Comparing the current of the parallelized code to the current of the serial code yields the error norm of **2.8776e-11**

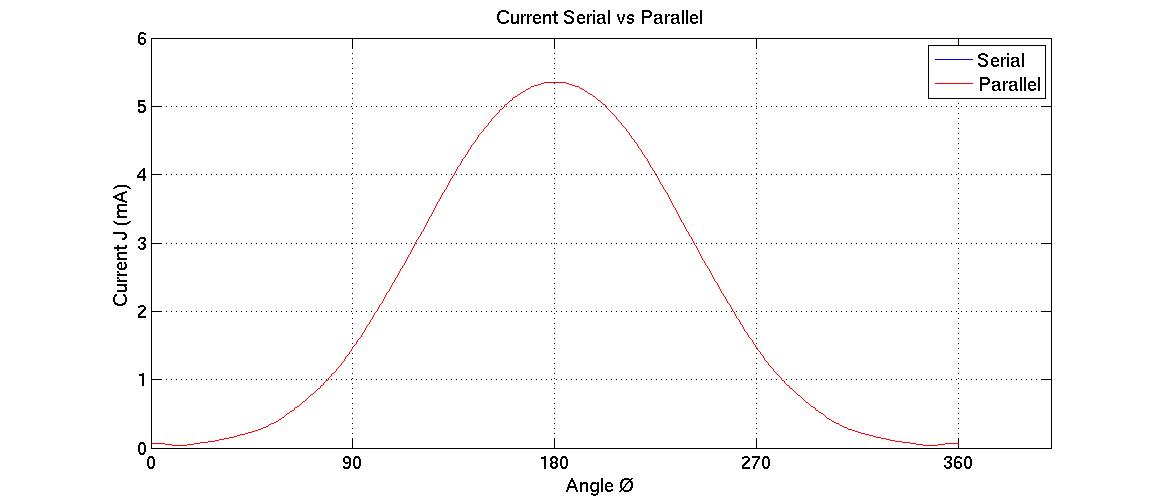


Figure Plot of current from Serial code, and Parallel code

* Comparing the scattered field of parallelized code to the scattered field of the Mie Series yields the error norm of **2.0e-3**, and comparing it to the scattered field of the serial code we get the error norm of **4.2202e-13**

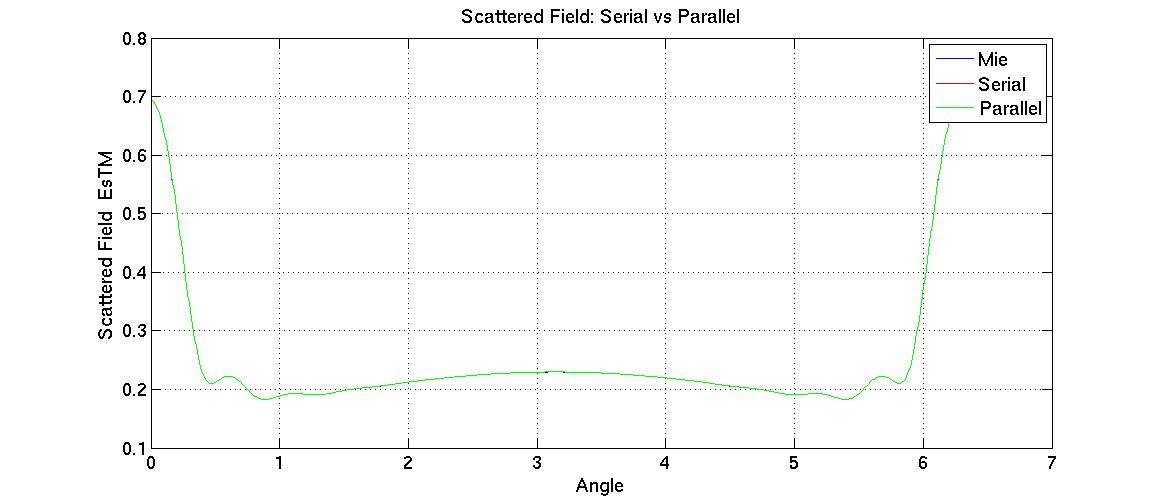


Figure Plot of current from serial code, and Parallel code

* The run time of the serial code is **0.110982** seconds. Comparing run time of the parallelized code based on number of processes used yields the following plot

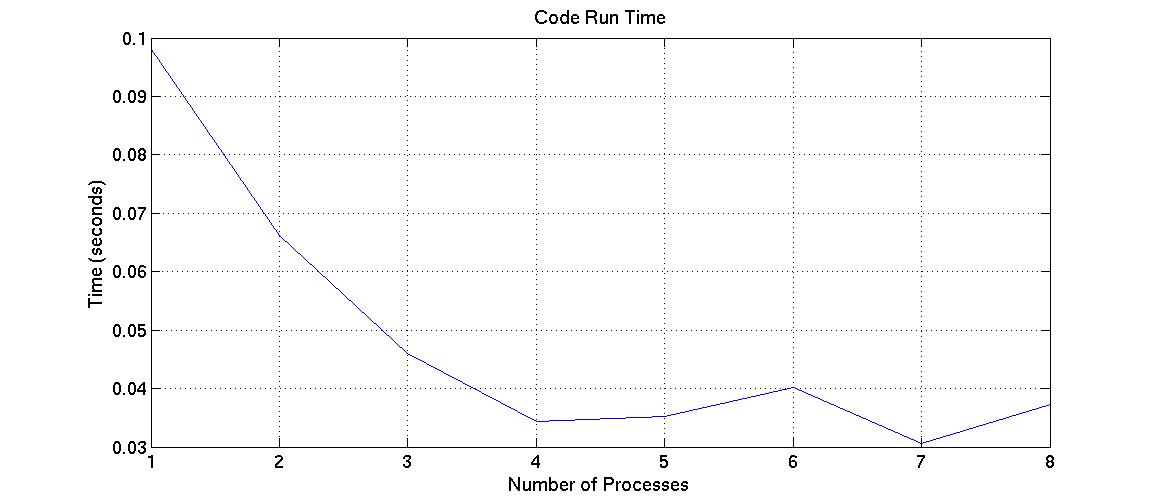


Figure Plot of runtimes of code based on number of processes used

**4 Conclusion**

In general, the results show great accuracy in calculations done in faster time; an optimized solution. Although differences exist between the currents calculated from the original code, the Mie Series, the LU method and the TFQMR method, Figures 1 and 4 show how minimal that difference is, and how similar the results are. Similarly, Figures 2 and 5 compare the results of the scattered fields, also with great accuracy.

The TFQMR method, and Open MPI have helped achieve the efficiency of the final products. The accuracy of the program was a direct result of manipulating the error tolerance of the TFQMR method. With less error tolerance, the iterative method produced more accurate results (Figure 3). The speed of the program is due its parallelization. Figure 8 shows the first trial of each run of the program with different amount of processes. The runtime is not constant, and is affected by the background activity of the computer, which explains the anomalies seen in Figure 8.

The results of parallelizing the code are astonishing. The results from the parallel code are accurate. When comparing the current with that of the serial code, it yielded the norm error of 2.8776e-11, and when comparing the scattered field with that of the serial code, it yielded the norm error of 4.2202e-13. The runtime of the code is faster than that of the serial code. The serial code executes in 0.110982 seconds, while the parallel code always executes in less time.

**5 Acknowledgements**

Special thanks goes to Professor Bagci for taking me under his mentorship and providing me with this great opportunity. I would also like to thank the PhD student whom I’ve worked with. Ali Imran provided me with great guidance; I wouldn’t have been able to finish this project without him. Sadeed Bin Sayed also provided me with great guidance; his technical virtuosity inspires me.

Special thanks also goes to KAUST for allowing me to work at and use its facilities and labs, and also for providing me with accommodation through my stay here. This whole experience has been made all the worthwhile with the support of my friends and family. My gratitude for them is undiminishing.

**References**

[1] Chew, Weng Cho, Mei Song Tong, and Bin Hu. *Integral Equation Methods for Electromagnetic and Elastic Waves*. United States of America: Morgan & Claypool, 2009. Print.

[2] Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with Message-Passing Interface second edition*. Cambridge: The MIT Press, 1999. Print.

[3] Anderson, E. and Bai, Z. and Bischof, C. and Blackford, S. and Demmel, J. and Dongarra, J. and Du Croz, J. and Greenbaum, A. and Hammarling, S. and McKenney, A. and Sorensen, D. *LAPACK: Users’ Guide*. 3rd ed .Philadelphia: Society for Industrial and Applied Mathematics, 1999. Print.

[4] **MATLAB and Statistics Toolbox Release 2012b, The MathWorks, Inc., Natick, Massachusetts, United States.**

[5] Moreau, Jean-Pierre. *TBESSY.F90* (Version 1.0) [source code]. Adapted From http://jean-pierre.moreau.pagesperso-orange.fr/Fortran/tbessy\_f90.txt

Appendix

1. Main Program

program MOMimplementation  
 use BesselH     !Bessel Functions Module  
 use global      !Variables and function  
 use MatVecMult  !Iterative method Module  
 use mpi   
 implicit none  
  
!Start allocations  
 allocate(theta(500), STAT = allocatestatus)  
 allocate(gx(size(theta)),STAT = allocatestatus)  
 allocate(gy(size(theta)),STAT = allocatestatus)  
 allocate(rm(size(gx)),STAT = allocatestatus)  
 allocate(theta\_m(size(theta)),STAT = allocatestatus)  
 allocate(Snm(size(gx),size(gx)),STAT = allocatestatus)  
 allocate(Phinm(size(gx),size(gx)),STAT = allocatestatus)  
 allocate(RHOmn(size(Snm,1),size(Snm,2)),STAT = allocatestatus)  
 allocate(ro(46,1),STAT = allocatestatus)  
 allocate(ph(1,700), STAT = allocatestatus)  
 allocate(X(size(ro),size(ph)),STAT = allocatestatus)  
 allocate(Y(size(ro),size(ph)),STAT = allocatestatus)  
 allocate(RHO(size(ro),size(ph)),STAT = allocatestatus)  
 allocate(PHI(size(ro),size(ph)),STAT = allocatestatus)  
 allocate(X\_n(size(RHO,1),size(RHO,2),size(gx)),STAT= allocatestatus)  
 allocate(Y\_n(size(RHO,1),size(RHO,2),size(gx)),STAT= allocatestatus)  
 allocate(RHO\_n(size(RHO,1),size(RHO,2),size(gx)),STAT = allocatestatus)  
 allocate(Hank\_TM(size(gx),size(gy)),STAT=allocatestatus)  
 allocate(precond(1:one\_n))  
 allocate(E\_i(L(1),L(2)))  
 allocate(Hank\_n\_TM(size(RHO\_n,1),size(RHO\_n,2),size(RHO\_n,3)))  
 allocate(ObsPhi(360))  
 allocate(RCS\_phi(size(ObsPhi)))  
!End allocations  
  
!!!!!!!!!Initializing Geometries!!!!!!!!!!  
  
 do i = 1,500  
    theta(i) = theta(i-1) + 2.0d0\*pi/500.0d0  
 end do  
  
 a      = .99d0\*lmda  
 k      = 2.0d0\*pi/lmda  
 Zo     = 120.0d0\*pi  
 sn     = a\*(theta(2)-theta(1))            !segment length  
 Factor = k\*sn\*Zo/4     
  
 one\_n = size(theta)  
  
   
 gx     = a\*cos(theta)                     !Grid on the scatterer (cylinder)  
 gy     = a\*sin(theta)  
  
!!!!!!!!!Scatterer discritization distance vectors!!!!!!!!!  
  
 rm = a   
 theta\_m = theta  
  
 Do m = 1, size(gx)  
    Do n = 1, size(gx)  
       Snm(m,n)   = sqrt(rm(m)\*\*2 + rm(n)\*\*2 - 2\*rm(m)\*rm(n)\*cos(theta\_m(n)-theta\_m(m)))   
       sy         = rm(n)\*sin(theta\_m(n)) - rm(m)\*sin(theta\_m(m))  
       sx         = rm(n)\*cos(theta\_m(n)) - rm(m)\*cos(theta\_m(m))   
       Phinm(n,m) = angle(sy,sx)   
    end do  
 end do  
  
 RHOmn = Snm    
  
! -----------------------------------------  
!   Grid Calculations  
!   Unique to both MoM and Mie Formulation  
! -----------------------------------------  
  
 do i = 1,46  
    ro(i,1) = 0.5d0 + 0.1d0\*(i-1)  
 end do  
  
 do i = 1,700  
    ph(1,i) = ph(1, i-1) + 2.0d0\*pi/700.0d0  
 end do  
  
 X = matmul(ro,cos(ph))  
 Y = matmul(ro,sin(ph))  
 RHO = sqrt(X\*\*2 + Y\*\*2)  
 PHI = atan2(Y,X)  
  
 r = size(RHO,1)  
 c = size(RHO,2)  
  
 L(1) = r  
 L(2) = c  
  
! -----------------------------------------------------------------  
!    Calculating Modified Grid |P - Pn|  
!    Distance of observation space w.r.t the object discritization  
! -----------------------------------------------------------------  
  
 do i = 1, size(gx) ! No of segments  
    do j = 1,L(1)    ! No of rows in the observation matrix RHO  
       X\_n(j,:,i) = X(j,:) - gx(i)  
       Y\_n(j,:,i) = Y(j,:) - gy(i)    ! Computed new P - P'  
    end do  
 end do  
 RHO\_n = sqrt(X\_n\*\*2 + Y\_n\*\*2)  
  
! --------------------------------  
! Calculation of Zmn  
! --------------------------------  
  
! --------------  
! # 2 - Appendix  
! --------------  
   
 do i = 1, size(gx)  
    do j = 1, size(gy)  
       Hank\_TM(i,j)  = bessel\_h(0.0d0, k\*abs(RHOmn(i,j)))  
    end do  
 end do  
  
   
 call MPI\_INIT(ierr)  
 call MPI\_COMM\_RANK(MPI\_COMM\_WORLD, rank, ierr)  
 call MPI\_COMM\_SIZE(MPI\_COMM\_WORLD, numprocs, ierr)  
 starttime = MPI\_WTIME()  
 print\*, "process number ", rank, " of ", numprocs  
 !----Settings----  
 ochunk = one\_n/numprocs  
 remainder = mod(one\_n,numprocs)  
 !----Determining Chunks(length) for each process----  
 chunk = ochunk  
 if (remainder /= 0 .and. rank <= remainder-1) chunk = ochunk + 1  
 !----Determining columns numbers for each process----  
 call segment(chunk,UB,LB)  
 print\*, "Process #", rank, "with LB,UB = ", LB, UB  
 print\*, "Processor num ", rank, 'with chunk', chunk  
  
 allocate(Zmn\_TM(size(gx),chunk),STAT = allocatestatus)  
 allocate(Vm(chunk), STAT = allocatestatus)  
 allocate(Jf(size(Vm)))  
 allocate(JfAll(one\_n),VmAll(one\_n))  
 allocate(pcount(numprocs),disp(numprocs))  
  
 do i = 1,size(gx)  
    do j = 1,chunk  
       if (i /= point(j)) then  
          Zmn\_TM(i,j) = Factor\*Hank\_TM(i,point(j))  
       else ! if (i == j) then  
          Zmn\_TM(i,j) = cmplx(Factor, -1\*Factor\*(2/pi)\*log(k\*Gama\*sn/(4\*e)))  
       end if  
    end do  
 end do  
 print\*, "Processor #",rank," finsihed filling Zmn\_TM"  
   
   
! -------------------------------  
! Calculation of Js  
! -------------------------------  
 do i=1,chunk  
 Vm(i)  = exp(cmplx(0,-1\*k\*a\*cos(theta(point(i))))) !incident field only on the scatterer  
 end do   
 print\*, "Processor #",rank," finsihed filling Vm"  
  
 !read input file to determine which method to use  
 open(unit=55,file='mom.inp',status='old')  
 read(55,\*) oper  
 close(unit=55)  
  
 if (oper == 1) then !Use LU method through Lapack subroutine ZGESV  
  
    JfAll = Vm  
    lda  = size(Zmn\_TM,1)  
    nn   = size(Zmn\_TM,2)  
    ldb  = size(Jf,1)  
    nrhs = 1    !size(Jf,2)  
    allocate(ipiv(nn))  
    call zgesv(nn,nrhs,Zmn\_TM,lda,ipiv,JfAll,ldb,info)  
      
 else if (oper == 2) then !Use Iterative method from MatVecMult module  
  
    disp=0  
    do i = 1, numprocs  
       if (i<=remainder)then  
          pcount(i)=ochunk+1  
       else  
          pcount(i)=ochunk  
       end if  
       disp(i)=disp(i-1)+pcount(i-1)  
    end do  
!!!!!gather RHS from different matrix sizes and distribute to all processes!!!!!  
    call MPI\_ALLGATHERv(Vm,chunk,MPI\_DOUBLE\_COMPLEX,VmAll,pcount,disp,MPI\_DOUBLE\_COMPLEX,MPI\_COMM\_WORLD,ierr)   
    JfAll=0   
    err   = 1e-12  
    iter  = 600  
    call initialize\_r0  
    call ztfqmr(one\_n,VmAll,JfAll,err,iter)   
    call MPI\_BARRIER(MPI\_COMM\_WORLD,ierr)  
  
 end if  
 endtime = MPI\_WTIME()  
 if(rank==0)print\*,'time is', endtime-starttime, 'seconds'  
 call MPI\_FINALIZE(ierr)  
    
! ---------------------------------------  
! Calculation of Scattered Field  
! ---------------------------------------  
  
 E\_i    = exp(cmplx(0,-1\*k\*RHO\*cos(PHI)))   !incident field on the observ. grid  
   
 do i = 1, size(RHO\_n,1)  
    do j = 1, size(RHO\_n,2)  
       do m = 1, size(RHO\_n,3)  
          Hank\_n\_TM(i,j,m)  = bessel\_h(0.0d0, k\*abs(RHO\_n(i,j,m)))  
       end do  
    end do  
 end do  
  
 allocate(Es\_TM(L(1),L(2)))  
 do i = 1, size(Hank\_n\_TM,3)  
    Es\_TM(:,:) = Es\_TM(:,:) - Factor\*Hank\_n\_TM(:,:,i)\*JfAll(i)  
 end do  
  
   
! ------------------------------------  
! Calculation of RCS  
! ------------------------------------  
  
 do i = 1, 360  
    ObsPhi(i) = ObsPhi(i-1) + 2\*pi/360  
 end do  
  
 Cl = 3e8  
 c\_j = cmplx(0.0d0,1.0d0)  
 do i = 1, size(ObsPhi)  
    RCS = 0  
    do j= 1, size(gx)  
       RCS = RCS + JfAll(j)\*sn\*exp(c\_j\*k\*(gx(j)\*cos(ObsPhi(i))+gy(j)\*&  
            sin(ObsPhi(i))))  
    end do  
    RCS\_phi(i) = (abs(RCS)\*\*2)\*(k/(4\*Cl\*\*2))  
 end do  
  
 print\*, 'reached final'  
  
!----Deallocate everything----  
deallocate(theta,gx,gy,rm,theta\_m,Snm,Phinm,RHOmn,ro,ph,X,Y,PHI,RHO,X\_n,Y\_n,RHO\_n,Hank\_TM,precond,E\_i,Hank\_n\_TM,Es\_TM,Vm,Jf,JfAll,VmAll,Zmn\_TM,r0\_initial,ObsPhi,RCS\_phi,pcount,disp)  
end program MOMimplementation

1. Global

module global   
implicit none  
  
 integer,parameter::dp=kind(0.0d0)  
 integer ::  i, j, m, n, allocatestatus  
  
 !---------- GEOMETRIES ----------  
 double precision,parameter:: lmda = 0.5d0, e = 2.718d0, Gama = 1.781d0, &  
      pi = 3.14159265358979323846264d0  
 double precision k, Zo, a, count, sn, Factor, sy, sx  
 integer          r, c, L(2)  
 double precision,Dimension(:)    ,allocatable::theta, gx, gy, rm, theta\_m  
 double precision,Dimension(:,:)  ,allocatable::Snm, Phinm, RHOmn,ro, &  
      ph, Y, X, RHO, PHI  
 double precision,Dimension(:,:,:),allocatable::Y\_n, X\_n, RHO\_n  
  
 !----------- CALCULATIONS -----------  
 complex(kind=dp),Dimension(:)    ,allocatable::Vm, Jf, JfAll,VmAll  
 complex(kind=dp),dimension(:,:)  ,allocatable::Hank\_TM, Zmn\_TM  
 complex(kind=dp),Dimension(:,:)  ,allocatable::E\_i, Es\_TM  
 complex(kind=dp),Dimension(:,:,:),allocatable::Hank\_n\_TM  
  
 !----------- LU METHOD VARIABLES -----------  
 integer,dimension(:),allocatable:: ipiv  
 integer :: info  
 integer :: nn, nrhs, lda, ldb  
  
 !----------- TFQMR METHOD VARIABLES -----------  
 complex(kind=dp),Dimension(:)    ,allocatable::r0\_initial, precond  
 integer one\_n, iter, oper  
 real(kind=dp) err  
 logical :: do\_precon = .False.  
  
 !----------- MPI VARIABLES -----------  
 integer :: ierr, rank, numprocs, remainder, chunk, LB, UB,ochunk  
 integer,dimension(:),allocatable::pcount,disp  
 double precision :: starttime, endtime  
  
 !----------- RCS VARIABLES -----------  
 double precision,Dimension(:),allocatable:: ObsPhi, RCS\_phi  
 double precision:: Wn, Cl   
 complex(kind=dp) :: RCS, c\_j,mm  
  
Contains  
  
  double precision function  angle(xComponent,yComponent)  
    double precision, INTENT(IN) :: xComponent, yComponent  
    double precision theta  
    theta = atan(yComponent/xComponent)  
    if (xComponent >= 0 .and. yComponent >= 0) then  
       angle = theta  
    else  
       if (xComponent < 0 .and. yComponent > 0) then  
          angle = pi - theta  
       else  
          if (xComponent < 0 .and. yComponent < 0) then  
             angle = pi + theta  
          else  
             angle = 1.5\*pi + theta  
          end if  
       end if  
    end if  
  end function angle  
  
  !!!!!!!!!!!Calculate the length, Lower Bound, and Upper Bound of each process!!!!!!!!!!  
  subroutine segment(chunk,UB,LB)  
    integer, intent(IN) ::chunk  
    integer, intent(OUT)::UB,LB  
    integer             ::factor  
    if (rank < remainder) then  
       LB = 1 + chunk\*rank  
       UB = chunk + chunk\*rank  
    else  
       LB = 1 + chunk\*rank + remainder  
       UB = chunk + chunk\*rank + remainder  
    end if  
  end subroutine segment  
  
  !!!!!!!!!!!Convert each column from local to global for calculations!!!!!!!!!  
  integer function point(uu)  
   integer, intent(IN) :: uu  
   point = LB+(uu-1)  
 end function point  
  
end module global

1. Bessel Function

Module BesselH  
use global   
IMPLICIT NONE  
CONTAINS  
  
complex function bessel\_h(N,X)  
double precision N, X  
if (N == 0) then  
  bessel\_h = cmplx(BESSJ0(X),-BESSY0(X))  
end if  
end function bessel\_h  
  
     double precision FUNCTION BESSY (N,X)  
! ------------------------------------------------------------------  
!     This subroutine calculates the second kind Bessel Function of  
!     integer order N, for any real X. We use here the classical  
!     recursive formula.   
! ------------------------------------------------------------------  
     double precision  X,TOX,BY,BYM,BYP,N,J   
     IF (N.EQ.0) THEN  
     BESSY = BESSY0(X)  
     RETURN  
     ENDIF  
     IF (N.EQ.1) THEN  
     BESSY = BESSY1(X)  
     RETURN  
     ENDIF  
     IF (X.EQ.0.) THEN  
     BESSY = -1.E30  
     RETURN  
     ENDIF  
     TOX = 2./X  
     BY  = BESSY1(X)  
     BYM = BESSY0(X)  
     DO 11 J = 1,N-1    
     BYP = J\*TOX\*BY-BYM  
     BYM = BY  
     BY  = BYP  
  11 CONTINUE  
     BESSY = BY  
     RETURN  
     END FUNCTION BESSY   
! ---------------------------------------------------------------------------  
     double precision FUNCTION BESSYP (N,X)   
     double precision  X,N   
     IF (N.EQ.0) THEN  
       BESSYP=-BESSY(1.0d0,X)  
     ELSE IF(X.EQ.0.D0) THEN  
       X=1.D-30  
     ELSE  
       BESSYP=BESSY(N-1,X)-(DBLE(N)/X)\*BESSY(N,X)  
      ENDIF  
     RETURN  
     END FUNCTION BESSYP  
! ---------------------------------------------------------------------------  
     double precision FUNCTION BESSY0 (X)   
     double precision  X,FS,FR,Z,FP,FQ,XX   
! ---------------------------------------------------------------------  
!     This subroutine calculates the Second Kind Bessel Function of  
!     order 0, for any real number X. The polynomial approximation by  
!     series of Chebyshev polynomials is used for 0<X<8 and 0<8/X<1.  
!     REFERENCES:  
!     M.ABRAMOWITZ,I.A.STEGUN, HANDBOOK OF MATHEMATICAL FUNCTIONS, 1965.  
!     C.W.CLENSHAW, NATIONAL PHYSICAL LABORATORY MATHEMATICAL TABLES,  
!     VOL.5, 1962.  
! ---------------------------------------------------------------------  
     double precision  Y,P1,P2,P3,P4,P5,R1,R2,R3,R4,R5,R6  &  
              ,Q1,Q2,Q3,Q4,Q5,S1,S2,S3,S4,S5,S6  
     DATA P1,P2,P3,P4,P5 /1.D0,-.1098628627D-2,.2734510407D-4, &  
     -.2073370639D-5,.2093887211D-6 /  
     DATA Q1,Q2,Q3,Q4,Q5 /-.1562499995D-1,.1430488765D-3,  &  
     -.6911147651D-5,.7621095161D-6,-.9349451520D-7 /  
     DATA R1,R2,R3,R4,R5,R6 /-2957821389.D0,7062834065.D0, &  
     -512359803.6D0,10879881.29D0,-86327.92757D0,228.4622733D0 /  
     DATA S1,S2,S3,S4,S5,S6 /40076544269.D0,745249964.8D0, &  
     7189466.438D0,47447.26470D0,226.1030244D0,1.D0 /  
     IF (X.EQ.0.D0) THEN  
     BESSY0 = -1.E30  
     RETURN  
     ENDIF  
     IF (X.LT.8.D0) THEN  
     Y = X\*X  
     FR = R1+Y\*(R2+Y\*(R3+Y\*(R4+Y\*(R5+Y\*R6))))  
     FS = S1+Y\*(S2+Y\*(S3+Y\*(S4+Y\*(S5+Y\*S6))))  
     BESSY0 = FR/FS+.636619772D0\*BESSJ0(X)\*LOG(X)  
     ELSE  
     Z = 8.D0/X  
     Y = Z\*Z  
     XX = X-.785398164D0  
     FP = P1+Y\*(P2+Y\*(P3+Y\*(P4+Y\*P5)))  
     FQ = Q1+Y\*(Q2+Y\*(Q3+Y\*(Q4+Y\*Q5)))  
     BESSY0 = SQRT(.636619772D0/X)\*(FP\*SIN(XX)+Z\*FQ\*COS(XX))  
     ENDIF  
     RETURN  
     END FUNCTION BESSY0  
! ---------------------------------------------------------------------------  
     double precision FUNCTION BESSY1 (X)   
     double precision  X,FR,FS,Z,FP,FQ,XX  
! ----------------------------------------------------------------------  
!     This subroutine calculates the Second Kind Bessel Function of  
!     order 1, for any real number X. The polynomial approximation by  
!     series of Chebyshev polynomials is used for 0<X<8 and 0<8/X<1.  
!     REFERENCES:  
!     M.ABRAMOWITZ,I.A.STEGUN, HANDBOOK OF MATHEMATICAL FUNCTIONS, 1965.  
!     C.W.CLENSHAW, NATIONAL PHYSICAL LABORATORY MATHEMATICAL TABLES,  
!     VOL.5, 1962.  
! ----------------------------------------------------------------------  
     double precision  Y,P1,P2,P3,P4,P5,R1,R2,R3,R4,R5,R6  &     
              ,Q1,Q2,Q3,Q4,Q5,S1,S2,S3,S4,S5,S6,S7  
     DATA P1,P2,P3,P4,P5 /1.D0,.183105D-2,-.3516396496D-4, &  
     .2457520174D-5,-.240337019D-6 /  
     DATA Q1,Q2,Q3,Q4,Q5 /.04687499995D0,-.2002690873D-3,  &  
     .8449199096D-5,-.88228987D-6,.105787412D-6 /  
     DATA R1,R2,R3,R4,R5,R6 /-.4900604943D13,.1275274390D13,   &  
     -.5153438139D11,.7349264551D9,-.4237922726D7,.8511937935D4 /  
     DATA S1,S2,S3,S4,S5,S6,S7 /.2499580570D14,.4244419664D12, &  
     .3733650367D10,.2245904002D8,.1020426050D6,.3549632885D3,1.D0 /  
     IF (X.EQ.0.) THEN  
     BESSY1 = -1.E30  
     ENDIF  
     IF (X.LT.8.) THEN  
     Y = X\*X  
     FR = R1+Y\*(R2+Y\*(R3+Y\*(R4+Y\*(R5+Y\*R6))))  
     FS = S1+Y\*(S2+Y\*(S3+Y\*(S4+Y\*(S5+Y\*(S6+Y\*S7)))))  
     BESSY1 = X\*(FR/FS)+.636619772\*(BESSJ1(X)\*LOG(X)-1./X)  
     ELSE  
     Z = 8./X  
     Y = Z\*Z  
     XX = X-2.356194491  
     FP = P1+Y\*(P2+Y\*(P3+Y\*(P4+Y\*P5)))  
     FQ = Q1+Y\*(Q2+Y\*(Q3+Y\*(Q4+Y\*Q5)))  
     BESSY1 = SQRT(.636619772/X)\*(SIN(XX)\*FP+Z\*COS(XX)\*FQ)  
     ENDIF  
     END FUNCTION BESSY1  
! ----------------------------------------------------------------------  
     double precision FUNCTION BESSJ0 (X)  
     double precision  X,AX,FR,FS,Z,FP,FQ,XX   
! ---------------------------------------------------------------------  
!     This subroutine calculates the First Kind Bessel Function of  
!     order 0, for any real number X. The polynomial approximation by  
!     series of Chebyshev polynomials is used for 0<X<8 and 0<8/X<1.  
!     REFERENCES:  
!     M.ABRAMOWITZ,I.A.STEGUN, HANDBOOK OF MATHEMATICAL FUNCTIONS, 1965.  
!     C.W.CLENSHAW, NATIONAL PHYSICAL LABORATORY MATHEMATICAL TABLES,  
!     VOL.5, 1962.  
! ---------------------------------------------------------------------  
     double precision  Y,P1,P2,P3,P4,P5,R1,R2,R3,R4,R5,R6  &    
              ,Q1,Q2,Q3,Q4,Q5,S1,S2,S3,S4,S5,S6  
     DATA P1,P2,P3,P4,P5 /1.D0,-.1098628627D-2,.2734510407D-4, &  
     -.2073370639D-5,.2093887211D-6 /  
     DATA Q1,Q2,Q3,Q4,Q5 /-.1562499995D-1,.1430488765D-3, &  
     -.6911147651D-5,.7621095161D-6,-.9349451520D-7 /  
     DATA R1,R2,R3,R4,R5,R6 /57568490574.D0,-13362590354.D0, &  
     651619640.7D0,-11214424.18D0,77392.33017D0,-184.9052456D0 /  
     DATA S1,S2,S3,S4,S5,S6 /57568490411.D0,1029532985.D0, &  
     9494680.718D0,59272.64853D0,267.8532712D0,1.D0 /  
     IF(X.EQ.0.D0) GO TO 1  
     AX = ABS (X)  
     IF (AX.LT.8.) THEN  
     Y = X\*X  
     FR = R1+Y\*(R2+Y\*(R3+Y\*(R4+Y\*(R5+Y\*R6))))  
     FS = S1+Y\*(S2+Y\*(S3+Y\*(S4+Y\*(S5+Y\*S6))))  
     BESSJ0 = FR/FS  
     ELSE  
     Z = 8./AX  
     Y = Z\*Z  
     XX = AX-.785398164  
     FP = P1+Y\*(P2+Y\*(P3+Y\*(P4+Y\*P5)))  
     FQ = Q1+Y\*(Q2+Y\*(Q3+Y\*(Q4+Y\*Q5)))  
     BESSJ0 = SQRT(.636619772/AX)\*(FP\*COS(XX)-Z\*FQ\*SIN(XX))  
     ENDIF  
     RETURN  
   1 BESSJ0 = 1.D0  
     RETURN  
     END FUNCTION BESSJ0  
! ---------------------------------------------------------------------------  
     double precision FUNCTION BESSJ1 (X)  
     double precision  X,AX,FR,FS,Z,FP,FQ,XX    
! ---------------------------------------------------------------------  
!     This subroutine calculates the First Kind Bessel Function of  
!     order 1, for any real number X. The polynomial approximation by  
!     series of Chebyshev polynomials is used for 0<X<8 and 0<8/X<1.  
!     REFERENCES:  
!     M.ABRAMOWITZ,I.A.STEGUN, HANDBOOK OF MATHEMATICAL FUNCTIONS, 1965.  
!     C.W.CLENSHAW, NATIONAL PHYSICAL LABORATORY MATHEMATICAL TABLES,  
!     VOL.5, 1962.  
! ---------------------------------------------------------------------  
     double precision  Y,P1,P2,P3,P4,P5,P6,R1,R2,R3,R4,R5,R6  &  
              ,Q1,Q2,Q3,Q4,Q5,S1,S2,S3,S4,S5,S6  
     DATA P1,P2,P3,P4,P5 /1.D0,.183105D-2,-.3516396496D-4,  &  
     .2457520174D-5,-.240337019D-6 /,P6 /.636619772D0 /  
     DATA Q1,Q2,Q3,Q4,Q5 /.04687499995D0,-.2002690873D-3,   &  
     .8449199096D-5,-.88228987D-6,.105787412D-6 /  
     DATA R1,R2,R3,R4,R5,R6 /72362614232.D0,-7895059235.D0, &   
     242396853.1D0,-2972611.439D0,15704.48260D0,-30.16036606D0 /  
     DATA S1,S2,S3,S4,S5,S6 /144725228442.D0,2300535178.D0, &  
     18583304.74D0,99447.43394D0,376.9991397D0,1.D0 /  
  
     AX = ABS(X)  
     IF (AX.LT.8.) THEN  
     Y = X\*X  
     FR = R1+Y\*(R2+Y\*(R3+Y\*(R4+Y\*(R5+Y\*R6))))  
     FS = S1+Y\*(S2+Y\*(S3+Y\*(S4+Y\*(S5+Y\*S6))))  
     BESSJ1 = X\*(FR/FS)  
     ELSE  
     Z = 8./AX  
     Y = Z\*Z  
     XX = AX-2.35619491  
     FP = P1+Y\*(P2+Y\*(P3+Y\*(P4+Y\*P5)))  
     FQ = Q1+Y\*(Q2+Y\*(Q3+Y\*(Q4+Y\*Q5)))  
     BESSJ1 = SQRT(P6/AX)\*(COS(XX)\*FP-Z\*SIN(XX)\*FQ)\*SIGN(S6,X)  
     ENDIF  
     RETURN  
     END FUNCTION BESSJ1     
  
end module BesselH

1. Iterative Method

module MatVecMult  
 !use global,only:dp,rank  
 !use global,only:one\_n,chunk  
 use global  
 use mpi   
 !use mpi  
 implicit none  
 save  
 !complex(kind=dp),allocatable::r0\_initial(:)  
contains  
!-----------------------------------------------------------------------------------------------------------------  
 subroutine ztfqmr(ntotal,b,x,err,iter)  
  ! b is RHS vector, x is the desired solution vector....J,,,  
   use global,only:r0\_initial  
   ! use mat\_vec\_mult,only:matvec,precon  
   implicit none  
   integer,intent(in)::ntotal  
   complex(kind=dp),intent(inout),dimension(1:ntotal)::x  
   complex(kind=dp),intent(in),dimension(1:ntotal)::b  
   complex(kind=dp),dimension(1:ntotal)::bb  
   real(kind=dp)::err,rerr  
   integer::iter,itmax,it  
   complex(kind=dp),dimension(1:ntotal)::w,yo,ayo,ye,aye,r,d,v  
   real(kind=dp)::ta,we,cm  
   complex(kind=dp)::etha,rho,amgis,ahpla,dum,beta  
   real(kind=dp)::bmag  
   integer::ndim  
   ! print\*,x  
   !stop  
   ndim=ntotal   
   itmax=iter  
   if (iter.eq.0) itmax=ntotal  
   bb(1:ndim)=precon(b(1:ndim))  
   !  
   !  set initial values  
   !  
   d(1:ndim)=cmplx(0.0\_dp,0.0\_dp,dp)  
   call matvec(x(1:ndim),r(1:ndim))  
   r(1:ndim)=bb(1:ndim)-r(1:ndim) !residual from the initial guess  
   w(1:ndim)=r(1:ndim)  
   yo(1:ndim)=r(1:ndim)  
   call matvec(yo(1:ndim),ayo(1:ndim))  
   v(1:ndim)=ayo(1:ndim)  
   we =0.0\_dp  
   etha=cmplx(0.0\_dp,0.0\_dp,dp)  
  
   ta=sqrt(dot\_product(r(1:ndim),r(1:ndim)))  
   rho=dot\_product(r0\_initial(1:ndim),r(1:ndim))  
   bmag=sqrt(dot\_product(bb(1:ndim),bb(1:ndim)))  
   rerr=ta/bmag  
     
   iters: do it=1,itmax  
      amgis=dot\_product(r0\_initial(1:ndim),v(1:ndim))  
      ahpla=rho/amgis  
      ye(1:ndim)=yo(1:ndim)-ahpla\*v(1:ndim)  
      call matvec(ye(1:ndim),aye(1:ndim))  
      !  start odd (2n-1) m loop  
      d(1:ndim)=yo(1:ndim)+(we\*we\*etha/ahpla)\*d(1:ndim)  
      w(1:ndim)=w(1:ndim)-ahpla\*ayo(1:ndim)  
      we=sqrt(abs(dot\_product(w(1:ndim),w(1:ndim))))/ta  
      cm=1.0d0/sqrt(1.0d0+we\*we)  
      ta=ta\*we\*cm  
      etha=ahpla\*cm\*cm  
      x(1:ndim)=x(1:ndim)+etha\*d(1:ndim)  
      !  check if the result has converged.  
      !a        if (err\*bmag .gt. ta\*sqrt(2.\*it)) then  
      !  
      !  start even (2n)  m loop  
      d(1:ndim)=ye(1:ndim)+(we\*we\*etha/ahpla)\*d(1:ndim)  
      w(1:ndim)=w(1:ndim)-ahpla\*aye(1:ndim)  
      we=sqrt(abs(dot\_product(w(1:ndim),w(1:ndim))))/ta  
      cm=1.0d0/sqrt(1.0d0+we\*we)  
      ta=ta\*we\*cm  
      etha=ahpla\*cm\*cm  
      x(1:ndim)=x(1:ndim)+etha\*d(1:ndim)  
      !  check if the result has converged.  
!---------------------------------------------------------------------------for  
!exporting fort 400..................  
      !if (mod(it,5)==0 .or. rerr<5.0\_dp\*err) then  
         call matvec(x(1:ndim),r(1:ndim))  
         r(1:ndim)=bb(1:ndim) -r(1:ndim)  
         rerr=sqrt(abs(dot\_product(r(1:ndim),r(1:ndim))))/bmag  
         if(rank==0)print\*,'#ofiter,error:',it,rerr   !annoying  
         write(4000,\*) it,rerr              ! writing the iteration  
!and residual error into a fort file  
         if (err > rerr) then  
            err=rerr  
            iter=it  
            return  
         endif  
      !end if  
! -------------------------------------------------------------------------------------------------------------  
      !  make preparations for next iteration  
      dum=dot\_product(r0\_initial(1:ndim),w(1:ndim))  
      beta=dum/rho  
      rho=dum  
      yo(1:ndim)=w(1:ndim)+beta\*ye(1:ndim)  
      call matvec(yo(1:ndim),ayo(1:ndim))  
      !MAGIC  
      v(1:ndim)=ayo(1:ndim)+beta\*( aye(1:ndim)+beta\*v(1:ndim) )  
   enddo iters  
   !  
   call matvec(x(1:ndim),r(1:ndim)) ! this is the imp thing to look for.,...  
   !MAGIC  
   r(1:ndim)=bb(1:ndim)-r(1:ndim)  
   err=sqrt(abs(dot\_product(r(1:ndim),r(1:ndim))))/bmag  
   iter=itmax  
   return  
 end subroutine ztfqmr  
! --------------------------------------------------------------------------------------------------------  
 subroutine matvec(xin,xout) ! takes input xin... any vector.... and   
!the subroutine already contains.. the mtrix... it multiplies the  
!vector with the Z matrix.... and gives output.... in xout....  
   use global,only:Zmn\_TM,ierr  
   implicit none  
   complex(kind=dp),intent(in)::xin(1:one\_n)     
   complex(kind=dp),intent(out)::xout(1:one\_n)   
   complex(kind=dp)::tempxout(1:one\_n)    
   integer::i,j  
  
   xout(1:one\_n)=cmplx(0.0\_dp,0.0\_dp,dp)  
   tempxout(1:one\_n)=cmplx(0.0\_dp,0.0\_dp,dp)  
  
  
      do j=1,chunk  
         tempxout(1:one\_n) = tempxout(1:one\_n) + Zmn\_TM(1:one\_n,j)\*xin(point(j))   
      enddo  
      !Collect xout and distribute to all processes for next iteration  
      call MPI\_ALLREDUCE(tempxout,xout,one\_n,MPI\_DOUBLE\_COMPLEX,MPI\_SUM,MPI\_COMM\_WORLD,ierr)   
  
 end subroutine matvec  
!-------------------------------------------------------------------------------------------------------------  
 subroutine initialize\_r0  
   use global,only:r0\_initial  
   implicit none  
   integer::jran,i  
   real(kind=dp)::r0\_dummy(1:2\*one\_n)  
  
   allocate(r0\_initial(1:one\_n))  
  
   ! jran=1211  
   jran=-3  
   do i=1,2\*one\_n  
      r0\_dummy(i)=2.0\_dp\*real(ran1(jran),dp)-1.0\_dp  
   end do  
   do i=1,one\_n  
      r0\_initial(i)=cmplx(r0\_dummy(i),r0\_dummy(one\_n+i),dp)  
   end do  
   print\*,'r0\_initial is filled...'  
   return  
  
 end subroutine initialize\_r0  
  
  
 function ran1(idum)  
   implicit none  
   integer,intent(inout)::idum  
   integer,PARAMETER::IM=2147483647,IQ=127773,IR=2836,IA=16807,NTAB=32  
   integer::NDIV  
   real,parameter::EPS=1.2E-7,RNMX=1.-EPS  
   real::am,ran1  
   integer::j,k,iv(NTAB)=0,iy=0  
   SAVE iv,iy  
  
   NDIV=1+(IM-1)/NTAB  
   AM=1./IM  
  
   if (idum<= 0 .or. iy== 0) then  
      idum=max(-idum,1)  
      do j=NTAB+8,1,-1  
         k=idum/IQ  
         idum=IA\*(idum-k\*IQ)-IR\*k  
         if (idum .lt. 0) idum=idum+IM  
         if (j .le. NTAB) iv(j)=idum  
      end do  
      iy=iv(1)  
   end if  
   k=idum/IQ  
   idum=IA\*(idum-k\*IQ)-IR\*k  
   if (idum .lt. 0) idum=idum+IM  
   j=1+iy/NDIV  
   iy=iv(j)  
   iv(j)=idum  
   ran1=min(AM\*iy,RNMX)  
   return  
 end function ran1  
  
! NO preconditioning  
 function precon(u)  
   use global,only:precond,do\_precon  
   implicit none  
   complex(kind=dp),intent(in)::u(1:one\_n)  
   complex(kind=dp)::precon(1:one\_n)  
   if (do\_precon) then  
      precon(1:one\_n)=precond(1:one\_n)\*u(1:one\_n)  
   else   
      precon(1:one\_n)=u(1:one\_n)  
   end if  
   return  
 end function precon  
  
end module MatVecMult

1. Input file

1 !type of operation (1=LU, 2=tfqmr)

1. Makefile

run: global.o TFQMRmod.o BesselH.o MoM\_Code\_PEC\_TM.o

mpif90 -o run global.o TFQMRmod.o BesselH.o MoM\_Code\_PEC\_TM.o -L. -llapack

global.mod: global.o global.f90

mpif90 -c global.f90

TFQMRmod.mod: TFQMRmod.f90 TFQMRmod.o

mpif90 -c TFQMRmod.f90

TFQMRmod.o: TFQMRmod.f90

mpif90 -c TFQMRmod.f90

BesselH.mod: BesselH.o BesselH.f90

mpif90 -c BesselH.f90

global.o: global.f90

mpif90 -c global.f90

BesselH.o: BesselH.f90

mpif90 -c BesselH.f90

MoM\_Code\_PEC\_TM.o: MoM\_Code\_PEC\_TM.f90

mpif90 -c MoM\_Code\_PEC\_TM.f90

clean:

rm global.mod besselh.mod matvecmult.mod global.o BesselH.o TFQMRmod.o MoM\_Code\_PEC\_TM.o run