FILE: CELL.F90

```
module cell

   use global_env

   integer, parameter :: cell_share_max=50

   integer :: cell_share_num
   integer, dimension(cell_share_max) :: cell_local_walkers

   integer, dimension(:,:,:), allocatable :: cell_list_root
   integer, dimension(:), allocatable :: cell_list

   integer, dimension(3) :: cell_size
   integer, dimension(3) :: n_cells
   integer, dimension(:,:), allocatable :: cell_walker

   contains

   subroutine cell_find_walkers(cell_index,cell_local_walkers)
      ! ****************************************
      ! Find all walkers in the specified cell
      ! ****************************************
      implicit none
      integer, dimension(:), intent(in) :: cell_index
      integer, dimension(:), intent(out) :: cell_local_walkers
      integer :: i, n

      i=cell_list_root(cell_index(X),cell_index(Y),cell_index(Z))

      if(i==0) then  ! no walker in the cell
         cell_local_walkers(1)=0
         return
      else ! found a walker in the cell
         n=1
         cell_local_walkers(n)=i
      end if

      do while(i>0)
         n=n+1
         i=cell_list(i)
         if(i==0) then  ! no more walker
            cell_local_walkers(n)=0
         else ! still more walkers
            cell_local_walkers(n)=i
         end if
      end do

   end subroutine cell_find_walkers

   function cell_coordinates(walker_pos)
      ! **************************************************
      ! Find the index of the cell the walker belongs to
      ! **************************************************
      implicit none

      integer, dimension(3) :: cell_coordinates

      integer, intent(in), dimension(3) :: walker_pos

      cell_coordinates = (walker_pos-1)/cell_size+1
```

```

end function cell_coordinates

subroutine cell_init(n_walkers,max_walkers,walker_pos)

   implicit none

   integer, intent(in) :: n_walkers, max_walkers
   integer, intent(in), dimension(:,:) :: walker_pos
   integer :: i

   if(any(cell_size<1) ) then
      write(fmain,*) '*** cell_init ***  cell_size is too small.'
      stop
   end if

   n_cells = box_size/cell_size+1

   if( any((n_cells-1)*cell_size /= box_size) ) then
      write(fmain,*) '*** cell_init ***  cell_size is not consistent.'
      stop
   end if

   allocate(cell_list(max_walkers))
   allocate(cell_walker(max_walkers,3))
   allocate(cell_list_root(n_cells(X),n_cells(Y),n_cells(Z)))

!*************************************
! Find a cell that an atom belongs to.
```

```
   !*************************************

      do i=1,n_walkers
         cell_walker(i,:) = cell_coordinates(walker_pos(i,:))
      end do


      ! Set initial cell lists.
      cell_list_root = 0
      cell_list = 0
      do i=1,n_walkers
         cell_list(i) = cell_list_root(cell_walker(i,X),
            cell_walker(i,Y),cell_walker(i,Z))
         cell_list_root(cell_walker(i,X),cell_walker(i,Y),
            cell_walker(i,Z)) = i
      end do

   end subroutine cell_init

! ****************
!  cell_crossing
! ****************
   subroutine cell_crossing(walker_id,walker_pos)

      implicit none

      integer, intent(in) :: walker_id
      integer, intent(in), dimension(3) :: walker_pos
      integer :: n
      integer, dimension(3) :: new_cell, old_cell

      new_cell = cell_coordinates(walker_pos)

      if(all(new_cell == cell_walker(walker_id,:))) return

      old_cell = cell_walker(walker_id,:)
      cell_walker(walker_id,:) = new_cell

      n = cell_list_root(old_cell(X),old_cell(Y),old_cell(Z))

      if(n<1) then
         write(*,*) 'Error in cell_list_root'
         stop
      end if

      if(n==walker_id) then
         cell_list_root(old_cell(X),old_cell(Y),old_cell(Z))=
            cell_list(n)
      else
         do while ( cell_list(n) /= walker_id )
           n = cell_list(n)
         end do
         cell_list(n) = cell_list(walker_id)
      endif
      cell_list(walker_id) = 0

      n = cell_list_root(new_cell(X),new_cell(Y),new_cell(Z))
      cell_list(walker_id) = n
      cell_list_root(new_cell(X),new_cell(Y),new_cell(Z))=
         walker_id

   end subroutine
```

```

subroutine cell_test(n_walkers)

   implicit none

   integer, intent(in) :: n_walkers
   integer :: n, n_errors
   integer :: i, j, k, l

   write(fdbg,*) '*** Checking consistency of cell assignment ***'
   write(fdbg,*)
   n=0
   n_errors=0
   do i=1,n_cells(Z)
      do j=1,n_cells(Y)
         do k=1,n_cells(X)
            l=cell_list_root(k,j,i)

            do while(l>0)
               n=n+1
               if(any(cell_walker(l,:)/=(/k,j,i/)))  then
                  write(fdbg, '(a,a,i6,a,3i4,a,3i4,a)') &
                  'cell mismatch found:',l,'(',cell_walker(l,:),') != (',k,j,i,')'
                  n_errors=n_errors+1
               endif
               l=cell_list(l)
            end do

         end do
         end do
      end do

      if(n<n_walkers) then
         write(fdbg,'(a,i6)')  'Some  walkers are missing: ',n
      else if (n>n_walkers) then
         write(fdbg,'(a,i6)')  'Too many walkers are found:',n
      else
         write(fdbg,'(a,i6)')  'All walkers are accounted:',n
      end if

      if(n_errors>0) then
         write(fdbg,*) 'Inconsistency is found.'
      else
         write(fdbg,*) 'No inconsistency is found.'
      end if
   end subroutine cell_test

end module cell
```