无网格计算方法的网格搜索方法

周吕文

2010 年 11 月 3 日

本文首先将对 Linked-list 和 data locality 这两种搜索网格中的粒子的方法及其实现作一定的介绍。Linked-list 和 data locality 方法广泛应用于无网格的计算中, 如分子动力学,SPH 及 DPD。这两种方法用巧妙的用数组对各网格中的粒子进行区分和索引，以达到能够很直接找到所需网格中所有粒子的索引，而不必让计算机花费更多的时间去计算和判断。

然后介绍半邻居搜索的原理，半邻剧居搜索是指每个粒子只需要搜索它周围一半的网格及自身所在网格中的其它粒子，并判断是否存在相互作用。

最后结合 Linked-list 方法和半邻居搜索，给出整个搜索过程。结合 data locality 方法和半邻居搜索方法的搜索过程与 Linked-list 方法类似，本文不再详述。

1 Linked-list cells 方法

Linked-list cells 方法以不规则存储模式来存取每一个粒子的相关信息. 以图 1粒子布为例, 介绍 Linked-list cells 获取每一个格子 (Cell) 中粒子的方法.

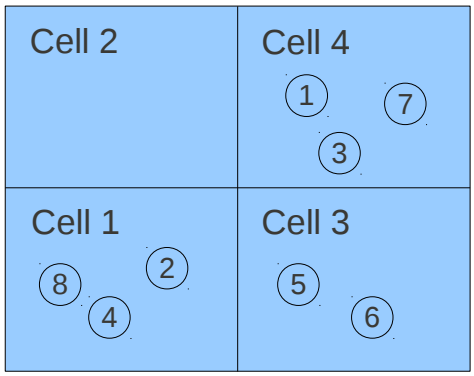

图 1: 各粒子所在网格分布示意图

该方法用 Head 和 Link 两个一维数组来存储网格和粒子的信息，图 2是对应图 1的 Head 和 Link 数组. Head 中第 i 个元素为第 i 个格子 (Cell i) 中的最大编号粒子的编号，比如第 3 个格子 (Cell 3) 中, 最大编号的粒子为 7, 因此 Head 中的第 3 个元素为 7. Link 中第 i 个元素表示与 i 号粒子在同一格子中且比 i 号粒子编号稍小的一个粒子的编号, 比如 Link 中第 4 个元素是 2，表示与 4 号粒子在同一格子中且编号比 4 号稍少的粒子是 2 号粒子。

	1	2	3	4
Head	8	0	6	7

	1	2	3	4	5	6	7	8
Link	0	0	1	2	0	5	3	4

图 2: Head 和 Link 数组

这样，我们就可以通过 Head 和 Link 这两个数组来访问某个格子中所有的粒子编号了，比如我们要访问第 4 个格子中的所有粒子, 则具体过程如下:

1. 先由 Head(4) = 7 找到第 4 个格子中最大号粒子即 7 号粒子
2. 再由 Link(7) = 3 找到第 4 个格子中次大号粒子即 3 号粒子
3. 再由 Link(3) = 1 找到第 4 个格子中次大号粒子即 1 号粒子
4. 再由 Link(1) = 0 判断第 4 个格子中没有其它可搜索的粒子

1.1 Linked-list cells 方法的实现

要应用 Linked-list cells 方法, 首先得根据粒子和网格的信息生成相应的 Head 和 Link 数组. 图 3是生成 Link 和 Head 数组的伪代码.

```
Algorithm 1: Generate Head and Link arrays
Initialize Link and Head
Do for all particles i form 1 to n
      Icell = The number of cell which contain particle i

      ! Link to the previous occupant
      Link(i) = Head(Icell)

      ! The last one goes to the header
      Head(Icell) = i
End do
```

图 3: 生成 Link 和 Head 数组的伪代码

由 Algorithm 1 生成 Link 和 Head 数组后, 就可以很方便的访问某个格子中的所有粒子, 实现访问的伪代码如图 4

```
Algorithm 2: Access All Particle in One Cell
i = Head(Number of one cell)
Do while I not equal 0
      Access the i-th particle
      i = Link(i)
End do
```

图 4: 访问某一格子中粒子的伪代码

2 Data locality 方法

Data locality 是一种按照常规的顺序存储各格子中的粒子信息. 如图 5, 该方法首先需要将所用粒子的序号 OldList 按照所在格子分好类, 生成新的序列为 NewList. 再用 CellEnd 数组存放每个格子中所有粒子在 NewList 中的位置, NewList 中第 CellEnd(i)+1 到第 CellEnd(i+1) 个元素表示第 i 个格子中的所用粒子. 比如 CellEnd(3)+1=4, CellEnd(3+1)=5, 表示 NewList 中第 4 到第 5 个元素表示第 3 个格子中的粒子序号.

	1	2	3	4	5	6	7	8
OldList	1	2	3	4	5	6	7	8
	1	2	3	4	5	6	7	8
NewList	2	4	8	5	6	1	3	7
	1	2	3	4	5			
CellEnd	0	3	3	5	8			

图 5: NewList 和 CellEnd 一维数组

2.1 Data locality 方法的实现

要应用 Data locality 方法, 首先得根据原粒子列表 OldList 和所在网格的信息生成相应的 NewList 和 CellEnd 数组. 图 6是生成 NewList 和 CellEnd 数组的伪代码. 附录 A 给出了对应图 8生成的 NewList 和 CellEnd 数组 Fortran95 的程序。

Algorithm 3: Generate NewList and CellEnd arrays

```
Initialize NewList, CellEnd and Temp

! Generate CellEnd array
Do for all particles i form 1 to n
      Icell = The index of cell which contain particle i
      CellEnd(Icell+1:end) = CellEnd(Icell+1:end) +1
End do

! Generate NewList array
Temp = CellEnd
Do for all particles i from n to 1
      Icell = The index of cell which contain particle i
      NewList( Temp(Icell+1) ) = I
      Temp(Icell+1) = Temp(Icell+1) - 1
End do
```

图 6: 生成 NewList 和 CellEnd 数组的伪代码

由 Algorithm 1 生成 Link 和 Head 数组后, 就可以很方便的访问某个格子中的所有粒子, 实现的访问伪代码如图 7.

Algorithm 4: Access All Particle in One Cell
Do for i form CellEnd(c)+1 to CellEnd(c+1) access particle NewList(i) End do

图 7: 访问某一格子中粒子的伪代码

3 半邻居网格搜索

首先我们将粒子所在区域划上网格, 如图 8. 并对网格进行编号。网格的编号索引有两种, 一种是用行列来表示，另一种是用单个下标来索引, 如红色粒子所在网格即可表示为 02 行 02 列, 也可用 06 号网格来引用.

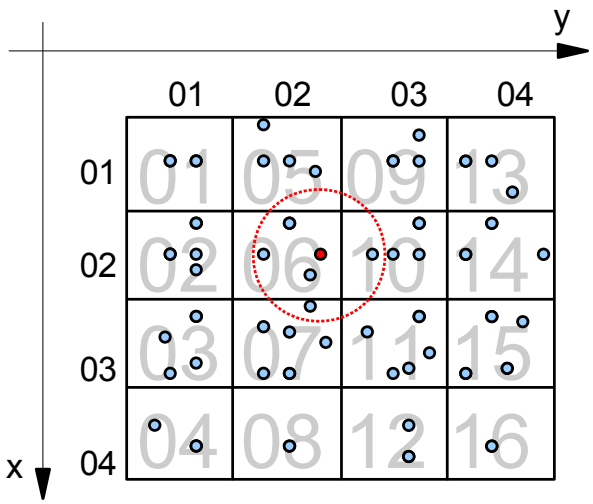

图 8: 二维空间网格划分示意图

我们假设有个 n 粒子, 第 i 个粒子空间坐标为 $r_i = [x_i, y_i]$, 将所有粒子的坐标存放在以下数组中:

$$r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

设空间的大小为 $boxl = [L_x, L_y]$; 将空间分成 $Nc_x \times Nc_y$ 个网格, 记 $N_c = [Nc_x, Nc_y]$, 则第 i 个粒子所在的网格 $CellSub_i = [CellSubX_i, CellSubY_j]$ 为

$$CellSubX_i = \lfloor \frac{x_i}{Nc_x} \cdot L_x \rfloor + 1, CellSubY_i = \lfloor \frac{y_i}{Nc_y} \cdot L_y \rfloor + 1$$

其中 $\lfloor x \rfloor$ 表示取小于等于 x 的最大整数, 通过下式，我们可以将行列索引转化为单个索引

$$CellInd_i = CellSubX_i + (CellSubY_i - 1) \cdot Nc_x = [CellSubX_i, CellSubY_i - 1] \begin{bmatrix} 1 \\ Nc_x \end{bmatrix}$$

至此, 我们可以由每个粒子的坐标位置得到该粒子所在的网格.

由于力的作用是相互的, 两个相互作用的粒子间的力是同时的, 这使得我们只需要搜索每一个粒子周围一半格子中的粒子, 并计算相互作用力. 图 9是这种搜索一半网格的示意图, 左图为二维情况, 只搜索周围 8 个邻居中的 4 个及自身, 即图中有色的 5 个格子; 右图为三维情况, 只搜索周维 26 个邻居中的 13 个及自身, 即图中画出的格子.

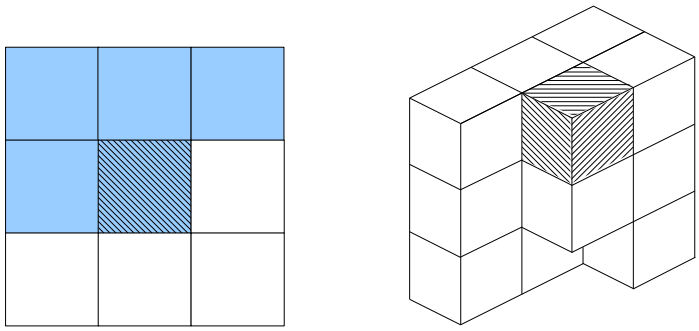

图 9: 搜索周围的一半网格示意图 (左: 二维, 右: 三维)

结合 Linked-list cells 和 data locality 方法, 我们可以完成搜索并计算出各粒子间的相互作用力, 图 10给出了结合 Linked-list cells 方法的搜索过程的伪代码. 具体程序见附录 B

Algorithm 5: half neighbors search base on Linked-list Cells (3D)

```
Do for all cells i
    CellInd_i = i
    atom_i = Head(CellInd_i)
    Do while atom_i not equal 0
        Do for all 14 cells j in the neighborhood of cell i
            CellInd_j = Index of cells j
            If CellInd_i equal CellInd_j then
                atom_j = Link(atom_i)
            Else
                atom_j = head(CellInd_j)
            End if
            Do while atom_j not equal 0
                !*****************************************
                compute the force between atom_i and atom_j in here
                !*****************************************
                atom_j = link(atom_j)
            End do
        End do
        atom_i = Link(atom_i)
    End do
End do
```

图 10: 基于 Linked-list cells 方法的网格搜索伪代码

A Local Data 方法的 NewList&CellEnd 生成程序举例

```
Program LocalData
!
! Purpose:
!       To generate the NewList and CellEnd array by linked-list cells
!
! Record of revisions:
!       Date            Programmer              Description of change
!       ====            ==========              =====================
!    Nov 1, 2010        Zhou Lvwen                  Original Code
!

implicit none
integer, parameter :: Nc = 4            ! Number of cells
! Cell Index of all atoms or particles
integer, dimension(8) :: CellInd = [4, 1, 4, 1, 3, 3, 4, 1]
integer, dimension(Nc+1) :: CellEnd = 0 ! CellEnd array
integer, dimension(Nc+1) :: Temp = 0    ! A copy of CellEnd for generate NewList
integer, dimension(8) :: NewList = 0    ! NewList array
integer :: i                            ! Loop index
integer :: Icell                        ! Cell index of particle i

do i = 1, 8
   Icell = CellInd(i)
   CellEnd(Icell+1:) = CellEnd(Icell+1:) + 1
end do

Temp = CellEnd

do i = 8, 1, -1
   Icell = CellInd(i)
   NewList( CellEnd(Icell+1) ) = i
   Temp(Icell+1) = Temp(Icell+1) - 1
end do

write(*,'(A, 8(1X,I2), 1X, A)') ' NewList = [', NewList, ']'
write(*,'(A, 5(1X,I2), 1X, A)') ' CellEnd = [', CellEnd, ']'

End program LocalData
```

B 基于 Linked-list cells 的网格搜索程序举例

```
Program LinkedList
!
! Purpose:
!       To give An example about half neighbors search base on Linked
!       -list cells
!
! Record of revisions:
!       Date            Programmer              Description of change
!       ====            ==========              =====================
!    Oct 26, 2010       Zhou Lvwen                  Original Code
!


implicit none
integer, parameter :: natom = 100                       ! Number of atoms or particles
integer, dimension(3), parameter :: Nc = [3, 3, 3]    ! Number of cells in x, y and z dimension
real   , dimension(3), parameter :: boxl = [6, 6, 6]   ! Box longth in x, y, and z dimension
real   , dimension(natom,3) :: r                        ! Position of atoms
integer, dimension(natom,3) :: CellSub              ! Multiple subscripts of cells: atoms
integer, dimension(3) :: CellSub_i, CellSub_j       ! Multiple subscripts of cells: atom i,j
integer, dimension(Nc(1)*Nc(2)*Nc(3)) :: head  = 0      !
integer, dimension(natom) ::  link = 0                  !
integer :: i, j, k, l                                   ! Loop index
integer :: CellInd_i, CellInd_j                         ! Index of cell i,j
integer :: atom_i, atom_j                               ! Index of atom i,j
integer, dimension(14,3) :: Dc                          ! 14 Neighbors of cell(0,0,0)
integer :: sub2ind                                      ! Function: Linear index from
                                                        ! multiple subscripts


! Build neighbor cells list
l = 1
do i = -1, 1
   do j = -1, 1
     do k = -1, 1
        if ((i + j + k)>=0 .and. k >= 0) then
           Dc(l,:) = [i, j, k]
           l = l+1
        end if
     end do
   end do
end do

! Generate the random position of all atoms or particles
do i = 1, 3
   do j = 1, natom
      call random_number(r(j,i))
```

```
   end do
end do

! Calculate cell subscript values of all atoms or particles
do i = 1, 3
   r(:, i) = r(:, i) * boxl(i)
   CellSub(:,i) = ceiling( r(:,i)/boxl(i) * Nc(i))
end do

! Generate the link array and head array by linked-list cells
do i = 1, natom
   CellInd_i = sub2ind(Nc, CellSub(i,:))
   write(*,*) CellSub(i,:), CellInd_i
   ! Link to the previous occupant (or EMPTY if you're the 1st)
   link(i) = head(CellInd_i)

   ! The last one goes to the header
   head(CellInd_i) = i
end do



do i = 1, Nc(1)*Nc(2)*Nc(3)
   CellInd_i = i
   atom_i = head(CellInd_i)
   do while (atom_i /= 0)

      do j = 1, 14
         CellSub_j =  CellSub(atom_i,:) + Dc(j,:)

         ! Periodic boundary condition
         do k = 1, 3
            if (CellSub_j(k) <  1    ) CellSub_j(k) = CellSub_j(k) + Nc(k)
            if (CellSub_j(k) >  Nc(k)) CellSub_j(k) = CellSub_j(k) - Nc(k)
         end do

         CellInd_j = sub2ind(Nc, CellSub_j)

         if (CellInd_i == CellInd_j) then
            atom_j = link(atom_i)
         else
            atom_j = head(CellInd_j)
         end if

         do while(atom_j /= 0)
            !************************************************
            !compute the force between atom_i and atom_j in here
            !
            ! r_ij = r(atom_i,:) - r(atom_j,:)
```

```
            !
            !************************************************
           atom_j = link(atom_j)
         end do
      end do
      atom_i = link(atom_i)
   end do
end do


End program LinkedList

!*******************************************************************************

Integer Function sub2ind(siz,sub)
!
! Purpose:
!       To determine  the equivalent single index corresponding  to a
!       given set of subscript values.
!
!       siz =[siz_x, siz_y, siz_z]   sub = [sub_x, sub_y, sub_z]
!
!                                         /  sub_x  \
!       ind = [siz_x, siz_y-1, siz_z-1] * |  sub_y  |
!                                         \  sub_z  /
! Record of revisions:
!       Date            Programmer              Description of change
!       ====            ==========              =====================
!    Oct 26, 2010        Zhou Lvwen                  Original Code
!

implicit none

integer, intent(in), dimension(3) :: siz ! arrary size
integer, intent(in), dimension(3) :: sub ! subscript value


sub2ind = dot_product(sub-[0,1,1],[1, siz(1), siz(2)*siz(3)])

End function sub2ind
```