Master in HPC

Problem Sheet 5 - kd-tree

In this exercise we want to construct a kd-tree in two dimensions starting from the same set of N points (input file 'tree.dat') used in the quadtree problem.

Unlike in the quadtree, a key difference is that a kd-tree is memory balanced , that is the cells boundaries are dynamically adapted so that at each level of the iteration a splitting coordinate x_{cut} is found such that for the generic subset of N_s particles there are $i = 1, 2, ..N_s/2$ particles with $x_i < x_{cut}$ and the remaining half $N_s/2 + 1, ..., N$ with $x_i \geq x_{cut}$.

Although there are many elements in common with the quadtree construction, for clarity we describe here the construction of the pseudocode as in the previous problem.

To construct the kd-tree at each level of the iteration it is necessary to find the splitting root x_{cut} which divides the particle subset. To this end it is used a modified version of the splitting routine $select(k, n, arr)$, introduced in sect. 8.5 of the Numerical Recipe book. For a given real array arr of n elements this routine returns the value of the $k - th$ smallest element which is found in the array.

In the original version the imput array is rearranged, while here the modified version returns the list of the ordered elements in an auxiliary array, leaving the input array arr unmodified. A fortran version of the routine is available upon request.

For the set of N input points write a program which computes the kd-tree defined by the array $pointers\_of\_tree[2, root : root + NC]$, here NC is the number of cells of the tree (typically $NC \simeq N$) and $root = N + 1$ is the cell starting address.

Let us consider the square of side length L which encloses all of the points. The kdtree is constructed by calling the routine *makekdtree*. This routine initializes the tree coordinates and constructs the kd-tree with a call to the recursive routine $tree\_split(nblist, listbodies, ic, iax)$. On input, *tree_split* has an array *listbodies* with the list of the *nblist* particles which have coordinates comprised within the boundaries of the cell with address ic and the splitting direction is defined by the $iax + 1$ component.

With the first iteration the root square of side L is divided along the

$iax+1$ direction into two sub-quadrants, defined by the splitting coordinate x_{cut}. This root is such that along the direction $iax+1$ there are $nblist/2$ particles with $x < x_{cut}$ and the other $nblist/2$ with $x \geq x_{cut}$.

The number of points into each of the two subqadrants is Np_sub. According to the value of Np_sub for the sub-quadrant j; $j = 1,\ 2$, the integer value $inext$ of the array $pointers\_of\_tree[j, root] = inext$ takes the values :

$Np\_sub > 1$ - more than one particle in the sub-quadrant, the sub-cell needs to be further examined : $inext =$ address new cell $> root$.

$Np\_sub = 1$ - there is one particle in the sub-quadrant, : $inext = i$ address of the particle.

$Np\_sub = 0$ - there are no particles in the sub-quadrant, there is no need to open further the sub-quadrant : $inext = 0$.

At each iteration the subcells with more than one particle are opened by a call to the recursive routine $tree\_split$ and cycling the splitting direction.

The procedure ends when there are no particles left to examine.

Write a program which reads as input file 'tree.dat' and for these points make a call to the function $makekdtree$ which constructs the kd-tree defined by $pointers\_of\_tree[2, root : root + NC]$.

At the end write on a file $kdtreecell.dat$, for all the cells of the kd-tree, the four coordinates (bottom, top, left,right).

Finally write on a file $kdtree\_ord.dat$ the particles coordinates, ordered according to their Morton key-value. Compute the keys by setting $levorder = 10$.

Finally, as in the quadtree case, make a plots of the particles coordinates together with the cells of the tree.

Here are given the corresponding pseudocodes.

Algorithm 1 KD-tree test

1: **procedure** TREE STRUCTURE ▷

Global:

Require: Int $Np = 4096,\ ncells = 2 * Np,\ nbodcell = Np + ncells$
Require: int $ndim = 2,\ nsubcell = 2$
Require: real $pos[2, nbodcell],\ bottom[2, nbodcell],\ cellsize[nbodcell]$
Require: int $pointers\_of\_tree[nsubcell, nbodcell]$
Require: int $iback[2, Np]$
Require: int $subindex[Np],\ bodlist[Np],\ isubset[Np]$
Require: int $incells,\ root,\ N$
Require: real side, rmin, rsize

Local:

Require: int $ic,\ m,\ i,\ pcell$
Require: real $xcell[4],\ ycell[4]$
Require: real $conv\_to\_int$ ▷ integer conversion
Require: int $levkey$
Require: long int $key\_M,\ Morton\_2D$ ▷ Morton key

Begin

2: Open $tree.dat$ ▷ read data file
3: read $side,\ N$ ▷ read box side and number of points
4: **if** $N > Np$ **then**
5: print $N,\ Np$
6: STOP
7: **end if**
8: **for** $i \leftarrow 1, N$ **do**
9: read $pos[1, i],\ pos[2, i]$ ▷ particles positions are stored in $pos[1 : 2, i]$
10: **end for**
11: $rmin := 0$ ▷ set tree boundaries
12: $rsize := side$

```
    CALL makekdtree                                   ▷ call the tree function
    Open kdtreecell.dat                                          ▷ write cell file
    print incells
    for ic ← 1, incells do
        pcell := ic + root − 1
        xcell[1] := bottom[1, pcell]
        ycell[1] := bottom[2, pcell]
        xcell[2] := bottom[1, pcell] + cellsize[pcell]
        ycell[2] := bottom[2, pcell]
        xcell[3] := bottom[1, pcell] + cellsize[pcell]
        ycell[3] := bottom[2, pcell] + cellsize[pcell]
        xcell[4] := bottom[1, pcell]
        ycell[4] := bottom[2, pcell] + cellsize[pcell]
        print ic, (xcell[m], ycell[m], m = 1, 4)
    end for
                                                  ▷ now compute the Morton keys
    levkey = 10
    conv_to_int = 2^levkey/side
    for i ← 1, N do
        ix = conv_to_int ∗ pos[1, i]
        iy = conv_to_int ∗ pos[2, i]
        key_M[i] = Morton_2D(ix, ix, levkey)
    end for
    CALL sorti(key_M, subindex, N) ▷ this call returns in subindex the
  list of the particles ordered by the value of key_M
    Open kdtree_ord.dat                                          ▷ write particle file
    print side
    for m ← 1, N do
        i:=subindex[i]
        print pos[1, i], pos[2, i], key_M[i]
    end for
end procedure
```

1: **procedure** MAKEKDTREE
Local:
Require: int $nbodlist,\ iaxstart,\ k,\ i$

2: $incells := 1$ ▷ set up number of cells, at least the root cell
3: $root := Np + 1$ ▷ set tree address
4: $pointers\_of\_tree = 0$ ▷ initialize pointer
5: $cellsize = 0$ ▷ initialize cellsize, note : particles are defined with $cellsize = 0$
6: **for** $k \leftarrow 1, ndim$ **do** ▷ set up position of root cell
7: $pos[k, root] := rmin[k] + rsize/2$
8: $bottom[k, root] := rmin[k]$
9: $cellsize[k, root] = side$ ▷ set up root cellsize
10: **end for**
11: **for** $i \leftarrow 1, n$ **do** ▷ now initialize particle list
12: $bodlist[i] := i$ ▷ all particles need to be examined
13: **end for**
14: $nbodlist := N$ ▷ actual value of the size of the particle list
15: $iaxstart := 0$ ▷ start the splitting with the x-axis
16: $CALL\ tree\_split(nbodlist, bodlist, root, iaxstart)$ ▷ call the recursive splitting routine
17: **end procedure**

1: **procedure** TREE_SPLIT($nblist, listbodies, ic, iax$) ▷ find the splitting coordinate and open the new cells

Require: int $nblist$, iax, j, jn, i, k, $msplit$, $jsplit$, $ifirst$, $ilast$

Require: int $npars$, $indcell$, $iaxnow$, p, m, $nsubc$, icn, ic, $pbody$, $selecti$

Require: real $bottomsplit$, $cellsplit$, $upside$

2: $iaxup := iax + 1$ ▷ new split direction

3: $iaxnow := MOD(iaxup - 1, ndim) + 1$ ▷ cycle around $ndim$

4: **for** $i \leftarrow 1, nblist$ **do** ▷ prepare coordinate list

5: $j := listbodies[i]$ ▷ which particle ?

6: $postemp[i] := pos[iaxnow, j]$

7: $listj[i] := i$ ▷ list in order

8: $subindex[i] := listbodies[i]$

9: **end for**

10: $msplit := nblist/2$ ▷ split in half

11: $jn = selecti(msplit, nblist, postemp, listj)$ ▷ memory address of the splitting coordinate $xcut = postemp[jn]$ ▷

$i = 1, msplit \Longrightarrow postemp[j = listj[i]] <= xcut$ ▷

$i = msplit + 1, nblist \Longrightarrow postemp[j = listj[i]] > xcut$

▷ Now reorder the input list according to the splitting, note that $listj$ is not ordered below or above $msplit$

12: **for** $k \leftarrow 1, nblist$ **do**

13: $i := listj[k]$

14: $listbodies[k] := subindex[i]$

15: **end for**

16: $xcut := pos[iaxnow, listbodies[msplit]]$

17: $upside := bottom[iaxnow, ic] + cellsize[iaxnow, ic]$

18: **for** $jsplit \leftarrow 1, 2$ **do**

19: **if** $jsplit = 1$ **then** ▷ particles on the left

20: $ifirst := 1$

21: $ilast := msplit$

22: $bottomsplit := bottom[iaxnow, ic]$

23: $cellsplit := xcut$

24: **else if** $jsplit = 2$ **then** ▷ particles on the right

25: $ifirst := msplit + 1$

26: $ilast := nblist$

27: $bottomsplit := xcut$

28: $cellsplit := upside - xcut$

29: **end if**

```
30:         npars := ilast − ifirst + 1
31:         if npars > 1 then                                   ▷ open new cell
32:             incells := incells + 1
33:             indcell := incells + root − 1
34:             p := indcell
35:             for m ← 1, ndim do                              ▷ new cell boundaries
36:                 cellsize[m, p] := cellsize[m, ic]
37:                 bottom[m, p] := bottom[m, ic]
38:             end for
39:             bottom[iaxnow, p] := bottomsplit
40:             cellsize[iaxnow, p] := cellsplit
41:             pointers_of_tree[jsplit, ic] := p     ▷ add new cell to the tree
42:             nsubc := 0
43:             for i ← ifirst, ilast do                        ▷ prepare new list
44:                 nsubc := nsubc + 1
45:                 k := listbodies[i]
46:                 listj[nsubc] := k
47:             end for
48:             icn := p
49:             CALL tree_split(nsubc, listj, icn, iaxnow)      ▷ split the new
    cell
50:         else if npars == 1 then  ▷ end the tree- append the particle
51:             i := ilast
52:             pbody := listbodies[i]
53:             pointers_of_tree[jsplit, ic] := pbody
54:             iback[1, pbody] := jsplit            ▷ from which cell is pbody ?
55:             iback[2, pbody] := ic
56:         end if
57:     end for
58: end procedure
```