Master in HPC

Problem Sheet 2 - Morton order

Let us consider a square of side length L. Write a program which computes the coordinates $xpos[i]$ and $ypos[i]$ of a set of points $N = M \times M$. The points must be arranged into the square with a uniform spacing. However, the program must compute the final list of points hierarchically, that is the program must contain a recursive function which has as input the root lattice ($N = 4 = 2 \times 2$) of points and it returns the $4N$ points which fill the four sub-squares. The order in which the sub-quadrants are filled must be : left-bottom, right-bottom, left-top, right top.

The function must be recursive so that for a given recursion depth $levscan$ it proceeds until there are $4^{levscan}$ points. Finally write the final points together with the corresponding Morton (or $Z-$) key values and make a plot of the points with a line joining them. Verify that for each point $i > 1$ it is satisfied $key(i) > key(i-1)$. Compute the $Z-$keys up to the order $levmorton$.

Assume as input $L = 16, levscan = 4, levmorton = 8$.

Here are given the corresponding pseudocodes.

Algorithm 1 Morton test

procedure POINT LATTICE ▷
  Global:
Require: Int Np=4096
Require: real xpos[Np], ypos[Np]
Require: real quad[0:1,4] ▷ array of unit box coordinates
Require: real corner[0:1,4] ▷ array of coordinates of the unit box corners
Require: real side ▷ side length of the square
Require: int $npoints := 0$ ▷ initial number of points
Require: int $levgrid := 0$ ▷ recursion level
  Local:
Require: real xgrid[0:3], ygrid[0:3]
Require: int $levmorton := 8$ ▷ Z-curve order
Require: int $nsub := 4$ ▷ first subdivision
Require: int $iad := 0$ ▷ address index
Require: int key_i ▷ Z-key
Require: int ix,iy ▷ integerized coordinates
Require: real $\Delta := 1$ ▷ point spacing unit box
Require: real $H := 0.5$ ▷ lattice shift
Require: real cj ▷ floating conversion
  Begin
  for $j \leftarrow 0, 1$ **do**
    for $i \leftarrow 0, 1$ **do**
      $iad := iad + 1$ ▷ increment address index
      $quad[0, iad] := i * \Delta + H$ ▷ set the basic points in row order
      $quad[1, iad] := j * \Delta + H$
      $corner[0, iad] := i * \Delta$ ▷ set the corners
      $corner[1, iad] := j * \Delta$
    end for
  end for
  $side := 16$ ▷ set the side length of the square
  $levscan := 4$ ▷ set the final level
  if $4^{levscan} > Np$ **then**
    print levscan, Np
    STOP
  end if
  for $i \leftarrow 1, 4$ **do** ▷ map the unit square to the root square
    $xgrid[i - 1] := quad[0, i] * side/2$
    $ygrid[i - 1] := quad[1, i] * side/2$
  end for

```
    CALL makegrid_morton(xgrid, ygrid, nsub)          ▷ call the recursive function
    cj := 2^levmorton/side                             ▷ integer conversion
    print side
    for i ← 1, npoints do
        ix := xpos[i] ∗ cj
        iy := ypos[i] ∗ cj
        key_i = morton2D(ix, iy, levmorton)
        print i, xpos[i], ypos[i], key_i
    end for
end procedure
```

```
procedure MAKEGRID_MORTON(xgrid,ygrid,n)
    input real xgrid[0 : n − 1]ygrid[0 : n − 1]
    local real xswap[0 : n − 1]yswap[0 : n − 1]
    local real xsub[0 : 4 ∗ n − 1]ysub[0 : 4 ∗ n − 1]
    if levgrid + 1 >= levscan then
        return
    end if
    levgrid := levgrid + 1
    for i ← 0, n − 1 do                                ▷ reduce to one-half
        xswap[i] := xgrid[i]/2
        yswap[i] := ygrid[i]/2
    end for
    for isub ← 0, 3 do                                 ▷ scan the four sub-quadrants
        iad = isub ∗ n
        for i ← 0, n − 1 do      ▷ now add the points of the sub square to the sub-quadrant
            xsub[i + iad] := xswap[i] + corner[0, isub] ∗ side/2
            ysub[i + iad] := yswap[i] + corner[1, isub] ∗ side/2
        end for
    end for
    CALL makegrid_morton(xsub, ysub, 4 ∗ n)            ▷ repeat for the new 4*n points
    if levgrid + 1 = levscan then      ▷ end of the recursion copy to final arrays
        for m ← 1, 4 ∗ n do
            xpos[m] = xsub[m − 1]
            ypos[m] = ysub[m − 1]
        end for
        npoints = 4 ∗ n
    end if
end procedure
```

```
procedure MORTON2D(ix,iy,bits)                 ▷ compute the 2D Z-key
    input integer ix, iy, bits
    local logical bitx, bity
    local integer levkey, key
    key := 0
    levkey := bits − 1
    while levkey >= 0 do             ▷ proceed until all the levels are done
        bitx := BTEST(ix, levkey)                     ▷ bit levkey of ix ?
        bity := BTEST(iy, levkey)                     ▷ bit levkey of iy ?
        if bitx then
            key := IBSET(key, 0 + 2 ∗ levkey)            ▷ set bit of key
        end if
        if bity then
            key := IBSET(key, 1 + 2 ∗ levkey)
        end if
        levkey := levkey − 1
    end while
    return Morton2D := key
end procedure
```