Pseudocode 1

Pseudocode for the Method-1(), which is used for distributed computation of uncertainty-aware catchment delineation without spatial partitioning of the DEM.

```plaintext
Procedure Method-1(D, int nodes, noise parameters ns, points pourpoints, points rivernetwork)

If(isRootNode())
  For n=1..nodes ;; n nodes
    sendStartCommand(n, D, nodes, ns, pourpoints, rivernetwork);
  EndFor
Else
  receiveDataFromRoot(D, nodes, ns, pourpoints);  
EndElse

iters = 0;

set_rivernetwork(D, rivernetwork, -infinity); ;; set height of stream network to -inf

While(needMoreResults()) ;; cont. until we have all results (the root tells the worker nodes when to stop)
  If(isRootNode())
    For n=1..nodes ;; n nodes
      While(hasData())
        R = readLocalResultsFromNode(n); ;; checks if node n has sent
        combineToResults(uncompress(R)); ;; results for master to process
        iters = iters + 1;
      EndWhile
    EndFor
    If(iters >= ITERLIMIT)
      SendTerminateMessageToNodes(nodes); ;; tells worker nodes to stop
    EndIf
  EndIf

N = processConvolution(sizeof(D), ns); ;; create realisation from the DEM error model
N = N + D; ;; add the realisation to the DEM
N = removeSurfaceDepressions(N); ;; remove sinks
N = calculateFlowDirections(N); ;; calculate flow directions
R = calculateDrainageBasin(N, pourpoints); ;; delineate catchment
sendToMaster(compress(R)); ;; sends results to the master
EndWhile
```

Supplemental Information
Supplemental Information

Pseudocode 2

Pseudocode for the Method-2(), which is used for distributed computation of uncertainty-aware catchment delineation with spatial partitioning of the DEM.

```plaintext
Procedure Method-2(DEM D, int nodes, noise params ns, node local central node, point pour points, time time limit, int ITERNUM)

; A single node (local central node) receives DEM data: D

Local central node divides computations and data among the other nodes

(DEM Di, Area Ai) = initClusters(nodes, ns, D);

set_rivernetwork(Di, rivernetwork, -infinity);

filter = calculateFilterMask(Ni, kernel type, phi, var); ;; explained in Section 2.2

iters = 0;

T1 = Start thread: R1i = ProcessConvolution(Ai, filter, Di); ;; explained in Section 3.3.1
T2 = Start thread: R2i = DepressionRemoval(R1i); ;; explained in Section 3.3.2
T3 = Start thread: R3i = CatchmentDelineation(R2i, pour points); ;; explained in Section 3.3.3
T4 = Start thread: R4i = CollectResults(R3i, local central node, filename); ;; explained in Section 3.3.4

While((iters < ITERNUM || isLocalCentralNode()) && getTime() < time limit && isRunning(T1) && isRunning(T2) && isRunning(T3) && isRunning(T4))

;; executes threads

If(isLocalCentralNode())

While(receivedResult())

iters = iters + 1;

EndWhile

EndIf

EndWhile

Stop threads T1, T2, T3, T4;
```
Pseudocode 3

Pseudocode for the `ProcessConvolution()`, which performs the distributed process convolution.

```
1 Procedure ProcessConvolution(Area Ai, ConvolutionMask F, DEM Di)
2
3 ;; create spatially uncorrelated Gaussian random field (GRF) Ni
4 Ni = CreateGRF(Ai);
5
6 ;; extend Ni for process convolution
7 ;; F and Ni are (2X + 1) x (2Y + 1) and K x L
8
9 E =
   \[
   \begin{pmatrix}
   N_{11}^{(1)} & N_{12}^{(2)} & N_{13}^{(3)} \\
   N_{21}^{(4)} & N_{22}^{(5)} & N_{23}^{(6)} \\
   N_{31}^{(7)} & N_{32}^{(8)} & N_{33}^{(9)}
   \end{pmatrix}
   \]
10
11 ;; send data to neighbouring nodes
12 For Each (node j, j!=i)
13    Receive[Aj] from node j; ;; area of node j
14    Bj = Aj U {X or Y wide border around Aj};
15    Start Send cells \( N_i[ Bj \cap Ai ] \) to node j;
16 EndForEach
17
18 ;; receive data from neighbouring nodes
19 For Each (node j and \( N_k^{(x,y)} \))
20    Receive cells \( A_j \cap Area(N_k^{(x,y)} \cap Ai) \) from j into \( N_k^{(x,y)} \);
21 EndForEach
22
23 E = ConvolveWithFFT(F, E); ;; convolves F with E using FFT
24 Ni = E[Ai]; ;; cuts only valid part of E
25 Ni = Di + Ni; ;; adds DEM data Di to the error surface
26
27 WaitForSends(); ;; wait for finishing of all sends
```
Pseudocode 4

Pseudocode for the DepressionFilling(), which makes distributed removal of the surface depressions, when DEM is spatially partitioned.

```plaintext
Procedure DepressionFilling(subset of DEM Di)

PriorityQueue OPENi; ;; separate data structures for each node i
Array visited, spill;

visited[.] = False; ;; initialization of data structures

For Each cell b in Di and border of DEM Di
    spill[b] = Di[b];
    OPENi.push(spill[b], b);
    visited[b] = True;
EndForEach

While OPENi in any node has had cells during the last two iterations of the loop
    For(ITERLOOP times if OPENi is not empty)
        (height, p) = OPENi.popLowest();
        ;; checks if a lower path has reached the cell already
        If(visited[p] and spill[p] < height]
            Continue; ;; cell p is skipped
        EndIf
        ForEach(neighbour n of cell p)
            k = max{Di[n], height};
            If(n not in Di)
                BufferedSend (p, k, n, SENDLIMIT) to node j where n /\ Dj;
                Continue;
            EndIf
            If(visited[n] = False)
                visited[n] = True;
                spill[n] = k;
                OPENi.push(k, n);
            ElseIf(spill[n] > k)
                spill[n] = k;
                OPENi.push(k, n);
            EndIf
        EndForEach
    EndFor
EndWhile

FlushSends(); ;; forces sending of buffered msgs

While(Receive (p, height, n) pair) ;; receives head of drainage paths/cells
    k = max(Di[n], height);
    If(visited[n] and spill[n] < height)
        Continue; ;; skips
    EndIf
    spill[n] = k;
    OPENi.push(k, n);
EndWhile
```

Supplemental Information
Pseudocode 5

Pseudocode for the `CatchmentDelineation()`, which defines the cells located on the catchment area of the pre-defined pourpoint(s).

```
Procedure CatchmentDelineation(Array Fi, Set Pi)

; Fi – depressionless DEM, Pi - drainage points.
Array visited, delineation, upperDEM, bottomDEM, leftDEM, rightDEM;
visited[] = False;
delineation[] = False;

; send and receives one-cell-wide borders around assigned part of Fi DEM.
sendAndReceiveDEMBorders(Fi, upperDEM, bottomDEM, leftDEM, rightDEM);
upperArea = size(upperDEM); bottomArea = size(bottomDEM);
leftArea = size(leftDEM); rightArea = size(rightDEM);

While(some PinotEmpty() in any node has had cells during the latest 2 iterations of the loop)

For(ITERLOOP times if Pi is not empty)
    Point x = Pi.remove();
    If(visited[x] = True)
        Continue;
    visited[x] = True;
    For Each n, 0<||n-x||<2
        If(n not in Di)
            BufferedSend (n to node j, x E Dj, SENDLIMIT); Continue;
    EndIf
    ; is there flow from n to x?
    If(CanFlowTo(n,Fi,x) = False or visited[n] = True)
        Continue;
    Pi.insert(n);
EndForEach
EndFor
FlushSends(); ; forces sending of buffered msgs

While(Receive(z))
    If(visited[z] = False)
        Pi.insert(z);
EndWhile
EndWhile

; delineate cells with unvisited neighbours
; query unknown cells from other nodes
For Each x in Di
    N = {n | 0<||n-x||<2}
    If(visited[x] = True and for some y in N, visited[y] = False)
        delineation[x] = True;
EndIf
EndForEach
```
Pseudocode 6

Pseudocode for the `CanFlowTo()` function, which defines the local flow direction by taking into account the heights of the DEM subset in the neighbouring node.

```
Function CanFlowTo(node target, Array height, node source)

If(height[source] < height[target] ) ;; cannot flow to target in any case
  return False;

value steepestHeightDiff = height[target]; ;; the steepest height/delta

For(dy = -1 .. +1)
  For(dx = -1 .. +1)
    If(dy == 0 && dx == 0)
      Continue;
    
    value x0 = source.x + dx;
    value y0 = source.y + dy;

    If(area(height).inside(x0, y0))
      If(steepestHeightDiff > height[x0, y0])
        return False; ;; search for steeper flow direction
      EndIf
    Else ;; search for better flow direction from boundaries sent from neighbouring nodes
      If(upperArea.inside(x0, y0))
        If(steepestHeightDiff > upperDEM[x0, y0])
          return False;
      EndIf
      ElseIf(bottomArea.inside(x0, y0))
        If(steepestHeightDiff > bottomDEM[x0, y0])
          return False;
      EndIf
      ElseIf(leftArea.inside(x0, y0))
        If(steepestHeightDiff > leftDEM[x0, y0])
          return False;
      EndIf
      ElseIf(rightArea.inside(x0, y0))
        If(steepestHeightDiff > rightDEM[x0, y0])
          return False;
      EndIf
    EndIf
  EndFor
EndFor

return True; ;; flow from source->target is in the steepest direction.
```
Supplemental Information

Pseudocode 7

Pseudocode for the `CollectResults()` function, which collects the results from the worker nodes.

```plaintext
1. Procedure CollectResults(Array Ri, string rootNodeName, Array groups)
2. rank = GetRank();
3. hname = GetHostname();
4. If(hname = rootNodeName && hasLowestRankInThisMachine(rank))
5.   dataCollectHost = True;
6. Else
7.   dataCollectHost = False;
8. EndIf
9. If(dataCollectHost == True)
10. For Each group G in Groups
11.   Initialize empty Array R;; R covers whole DEM area (R is swapped to the disk and
12.      For Each node N in group G;; only a single node's area is in the memory at once.)
13.     Array temp(N.area);
14.     StartReceiveDataFrom(N.rank, temp);
15.     N.temp = temp;
16.   EndForEach
17. If(rank belongsTo group G)
18.   sendDataToDataCollectHost(compress(Ri), rootNodeName);
19. EndIf
20. WaitForReceivesToEnd();;; waits for all the receives to arrive from this group
21. ;; decompresses received data and combines the parts into a single drainage basin result
22. For Each node N in group G
23.   R(N.area) = decompress(N.temp);
24. EndForEach
25. addResultToTheFinalMeanResult(R);
26. EndForEach
27. Else
28. ;; blocks until the data collect node has received the data
29.   sendDataToDataCollectHost(compress(Ri), rootNodeName);
30. EndIf
```