Lecture 10b – Hybrid Data Structures for Multi-Block Solution Procedures

- **Unstructured-grid procedures have random node numbering for each cell**
  - This allows for cells to be created arbitrarily to fill in a domain and straightforward cell division for grid-adaptation schemes
  - However, it requires a data structure that has:

    nodes → edges
    edges → faces
    faces → cells
    neighbor cells → cells

    Note that these are essentially objects (classes)

  Thus the data structure between the nodes and the cells can be multi-level which leads to expensive indirect addressing of memory
- **We can greatly reduce this overhead by treating blocks in an unstructured grid format while keeping the cells inside of the block in a structured-grid format**
  - This is the concept of "multi-block" structured grids

Multi-Block Structured Grids

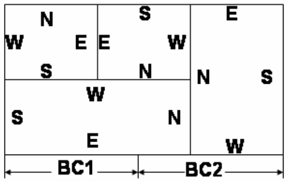
- **So the advantage of multi-block structured grids are:**
  - Practically any geometry can be gridded using unstructured-grid like block topologies that are connected using a block-neighbor data structure
    - The resulting data structure does not need all of the overhead of a totally unstructured-grid procedure
    - Only neighbor information is necessary since the nodes on adjacent faces of different blocks can be easily mapped
    - Lends very well to both distributed- and shared-memory parallel computer architectures
  - The operators inside of the block are all structured using direct addressing of memory and rapid indexing
    - Lends very well to vector and/or graphical processors
  - Grid adaptation can be performed uniformly and directionally on a per-block or per-sub-block basis

Multi-Block Structured Grids – Additional Considerations

- **Additional information must now be added to single-block solvers to allow for multi-block topologies:**
  - For each given block, store:
    - Global block number
    - Block type (in multi-disciplinary problems, "solid" or "fluid")
      - This could also include "overlaid", "structured", "unstructured"
    - Processor number (parallel) the block belongs to
    - Possibly the orientation
  - For each given block face, store:
    - The start and end indices (note that start index can be larger than end index to reverse order of processing)
    - The adjacent global block number (if one exists)
    - The face of the adjacent block (if one exists)
    - The orientation of the adjacent block OR the start and end indices in the directions of the adjacent block

Possibility of Sub-Faces

- **In many cases, you will find that it is inconvenient topology-wise to create blocks in such a fashion that the block faces have a single boundary condition over the entire face**
- **Multiple sub-faces can be used to allow for:**
  - Blocks to be staggered
  - Multiple physical boundary conditions to be applied along a face



- **This de-couples the block topology from the physical boundary conditions**
- **Sub-faces are not necessary for your projects. However, they are handy for general purpose procedures.**

Sub-Face Considerations

- **Incorporating sub-faces into a solver requires**
  - Number of sub-faces for each face (S,N,W,E,H,Y) as part of the stored block information
  - Face information:
    - Start and stop indices (in each direction) of the sub-face
      - Must have the capability to traverse in positive or negative i-, j-, k-directions (ie start index can be larger than stop index)
      - Note that traversing of face information MUST be consistent at inter-block boundaries since message passing has an order to it
    - Physical boundary condition type or
      - Along with any physical information regarding that boundary condition (eg total pressure, total temperature, etc)
    - Neighboring block number
      - Along with the start and stop indices (in each direction) that is point-matched to the given sub-face. Note that traverse direction of data in neighbor must be consistent with your own face traverse direction.

Blocks and Faces as Objects (Abstraction)

- **When coding up your projects, you could think of block faces abstractly (generically).**
  - You could create a (linked) list of faces that have
    - The list could consist of pointers to the temperature of the face nodes
    - Physical boundary conditions
    - Inter-block boundary treatments
  - For each face in the face list, you could store
    - Start and end point indices to loop over
    - For physical boundaries, the boundary condition type
    - For inter-block boundaries,
      - Neighbor block, face, and processor numbers
      - Indices of start and end points to loop over in each direction
- **You could also think of blocks abstractly (generically).**
  - You could create a (linked) list of blocks that have
    - Block type (for your projects, you only have solid blocks)
    - Processor number
    - Dimensions in each direction
    - Number of sub-faces in each direction

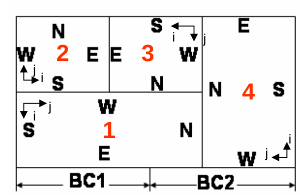
This is the unstructured-grid manner of processing blocks and faces

Blocks and Faces

- **<u>OR</u> you could simply loop over the south, north, west, and east faces**
  - Performing kernels for
    - Physical boundary conditions
    - Inter-block boundary conditions
  - This tends to produce repeatable sections of code, however, since the faces are doing similar tasks
- **<u>AND</u> loop over the blocks in a Cartesian grid system (since in your problem, the grid is broken up into N x M blocks)**
  - This is NOT typical, however, so the creation of a list of blocks is more general

Options for Orientation

- **Keeping track of block orientation is only important in terms of neighbor information to find out what neighbor face is adjacent to your face**
- **Block orientation can be performed either by**
  - Keeping track of how a given face maps to a standard-oriented block face
    - Let standard i- be 1 and standard j- be 2.
      - block 2 would have orientation of 1,2
      - block 1 would then have an orientation of -2,1.
      - block 3 would then have an orientation of -1,-2
      - block 4 would then have an orientation of 2,-1
    - BE CAREFUL! All blocks should be right-handed to make integrations come out correct.
  - OR Keeping track of start and end indices of each face (or sub-face) and matching them to neighbor correctly
    - Note that block 2's east face is ordered in the opposite direction as block 3's east face



TAGS for Send/Receiving Messages at Inter-Block Boundaries

- **Tags of Send and Receive Messages must be unique AND must be the same on the Send and Receive sides**
- **Combinatorial mathematics can be used to determine a unique TAG number:**
  - A combination of 2 positive, non-zero numbers A and B will be unique if A+BN is unique. For this to happen, N=max(A).
  - A combination of 3 positive, non-zero numbers A, B, and C will be unique if A+BN+CM is unique or when N=max(A) and M=max(A)+max(B)max(A).
  - A combination of 4 positive, non-zero numbers A, B, C, and D will be unique if A+BN+CM+DQ is unque or when N=max(A), M=max(A)+max(B)max(A), and

    Q=max(A)+max(B)max(A)+(max(A)+max(B)max(A))max(C) or Q=(max(A)+max(B)max(A))(1+max(C))
  - and so on...

TAGS

- **The number of unique tags available will depend on the number of variables that you use, A, B, C, D, etc.**
- **For your projects,**
  - A could be your block number
  - B could be your face number (S=1, N=2, W=3, E=4)
  - C could be your neighbor’s block number
  - D could be your neighbor’s face number that is adjacent to you
  - Etc.
- **There are other combination of variables that can be used to make up TAGS**
- **Try to keep the TAG formulas as simple as possible that give you the required number of unique values**