Lecture 9 – Spatial Domain Decomposition and Data Structures for Multi-Block Computations

- ***Domain Decomposition*** **has several meanings**
- **There are different types of domain decomposition:**
  - ***Spatial/grid domain decomposition*** is where a single domain is broken up into multiple "blocks" or sub-domains in order to
    - Make grid generation easier
    - Use parallel computing to solve larger problems and/or solve a problem faster
  - *Discipline domain decomposition* is where a domain is broken up into different physics (with different equation sets and/or models)
    - For different disciplines
    - For different physics
  - *Algorithm domain decomposition* is where different algorithms are used depending on purpose
    - Convergence acceleration (Eg multi-grid) vs fine-grid
  - ***Processor/Parallel block domain*** *decomposition* is where multiple blocks are distributed across multiple processors in such a way
    - Evenly distribute the work-load (load balancing) across the processors

Data Structures for Spatial/Grid and Processor/Parallel Domain Decomposition

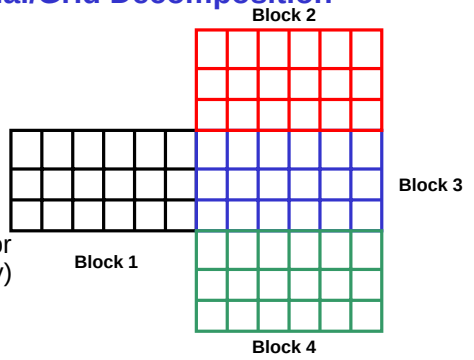
- **When developing a parallel code to solve an engineering simulation, we have to consider**
  - Where data must be exchanged at block and processor boundaries during the course of the algorithm
  - What data must be exchanged
  - If barriers are necessary to synchronize the processors
  - What blocks and processors must exchange data
- **Different engineering simulation codes may use different types of computational grid and differencing (or integration) stencil**
  - Multi-block structured grids
  - Multi-block unstructured grids
  - Multi-blocks of hybrid, structured/unstructured grids

Multi-Block Data Structures

- **Inter-block and inter-processor communication is dependent on block neighbor information**
- **Each block (of computational points or cells) must keep track of**
  - Its own block and processor number
  - Block and processor numbers of its neighbors (E, W, N, S, H, Y)
  - The boundary conditions to be applied along its face (edge)
  - Additional information for special applications (such as parent/child relationships for embedding-adaptation)
- **Upon execution, each block can then determine**
  - The processor which has been assigned to it
  - The processors associated with its neighbors
  - If communication is necessary for each block face (edge)

Example: Spatial/Grid Decomposition

- **Flow through a sudden expansion, or heat transfer in a "T" strut, etc.**
- **Each block must store**
  - Its N, S, W, E, H, Y neighbor block numbers (connectivity)
    - Example: Block-1 has neighbors 0,0,0,3,0,0
  - The boundary condition types (if any) at each point along is N, S, W, E, H, Y faces (edges)
    - Example: Block-1 has a BC type of 1 (solid boundary) along its southern boundary or –3 along its eastern boundary (which says that this is a block/block boundary



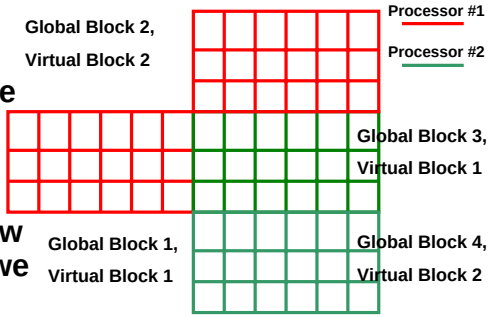
- Another way to store BC information would be to store regions of a face (sub-faces) with descriptions of the region dimensions and BC type
- Yet another way to store BC information would be point-by-point. This may make the user input description tedious, however.

Creation of Multi-Block Grids

- **The creation of a multi-block structured, unstructured, or hybrid grid is generally performed with a separate grid-generation code.**
  - A *multi-block restart file* can be written that contains the block dimensions, connectivity, and coordinates
    - PLOT3D is an "old" file format for restart files
    - CGNS is a new (modern) data-base format for restart files (you can find information regarding CGNS at: www.cgns.org
  - The initial values of the simulation code may be written in these same formats either as a separate file (PLOT3D) or as part of the data-base (CGNS)
- **In the case of your project simulation code, the next step will be to write a separate code to break up the original single-block grid into multiple blocks with connectivity and BC information (spatial/grid domain decomposition) for serial execution (projects 2 and 3).**
  - Note, however, that you will ultimately need to create the data structure for parallel execution (projects 4 and 5).

Example: Processor/Parallel Domain Decomposition

- **Processor/parallel domain decomposition may be used to determine the processor assignments to each block**
- **For instance, say we knew ahead of execution that we could use 2 processors**
  - Processor #1 is assigned to block 1 and 2
  - Processor #2 is assigned to block 3 and 4
- **So at this point, each processor can find out its neighboring processors via the block neighbor information**



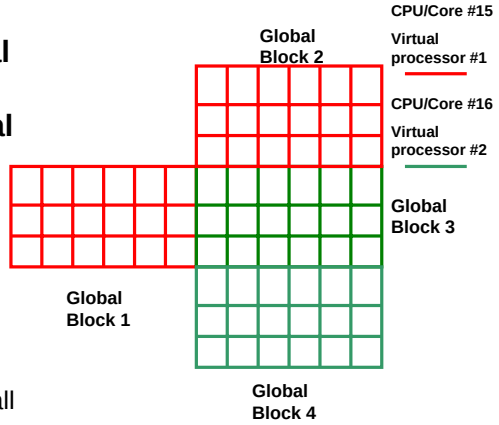
- **For instance, block-1:**
  - Knows that its east boundary is a block/block interface from its BC information for that face (edge)
  - Knows its east neighbor is block-3
  - Can determine that block-3's processor is #2 so the east face (edge) is also an inter-processor boundary

Example: Block and Processor Decomposition

- **So before execution even begins, we know the following for each processor:**
  - Block numbers of all blocks
  - Grid dimensions of all blocks
  - Coordinates and physical variables at nodes or cell-centers of all blocks
  - Boundary condition types for all points on all faces (edges) of all blocks
  - Neighbors of all blocks (N, S, W, E, H, Y)
  - Inter-processor boundaries (if any) for all blocks
  - The processor numbers assigned to the neighbors across inter-processor boundaries (This tells us who to communicate with)
- **Most of this information could be generated by a pre-processor spatial decomposition or grid-generation code, then written out to a file for each processor that could be read at the beginning of simulation execution.**
  - Determination of inter-processor boundaries and block assignments to virtual processors (last 2 steps above) would come later during processor/parallel domain decomposition

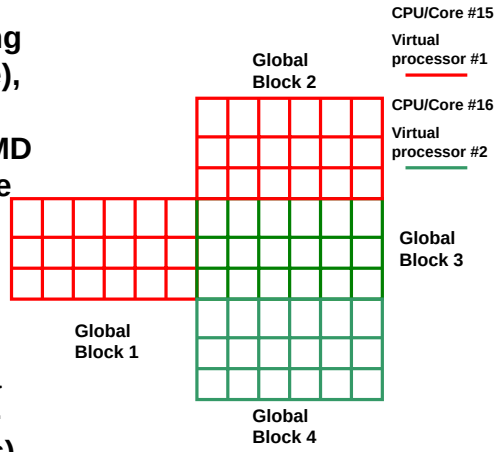
Example: Virtual to Physical Processor Mapping

- **Upon execution, physical processor numbers are mapped to computational processor numbers by MPI**
  - Example: CPU/Core #15 is assigned all the blocks of computational processor #1 (red blocks) and reads the pertinent block information
  - CPU/Core #16 is assigned all the blocks of computational processor #2 and reads all the pertinent block information (green blocks)
  - Solution iteration begins



Example: SPMD vs MPMD Parallel Computing

- **If all blocks are executing the same analysis (code), then this type of parallel computing is called SPMD (Single Process, Multiple Data)**
- **If the red blocks were executing one type of analysis and the green blocks were executing a different analysis (Multi-Disciplinary Simulations), then this type of parallel computing is called MPMD (Multiple Process,Multiple Data)**



Data Structures to Aid in Multi-Block Discretization and Parallel Communication

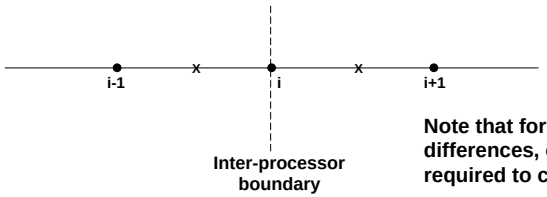
- **So far, we have only addressed issues related to breaking up the grid, assigning sub-groups of blocks to processors, and making the boundary condition specification more general**
- **We now must address the communication that takes place between blocks and processors during iteration of the solution algorithm**
- **This involves the discretization stencil which is dependent on the type of algorithm being used**
  - Finite difference, finite volume, finite element, other

Data Structures to Aid in Multi-Block Discretization and Parallel Communication

- **The differencing (or integration) stencil in engineering simulations depends on the governing equations to be solved and the numerical algorithm used to solve them**
  - For instance, most 2nd order-accurate difference operators used for the physical science simulations use a 3-point stencil

$$\left.\frac{\partial \phi}{\partial x}\right|_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \quad or \quad \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2}$$

  - One must keep the largest stencil used during the course of execution in mind when developing parallel code



Note that for typical 1st and 2nd differences, one adjacent point is required to create the stencil

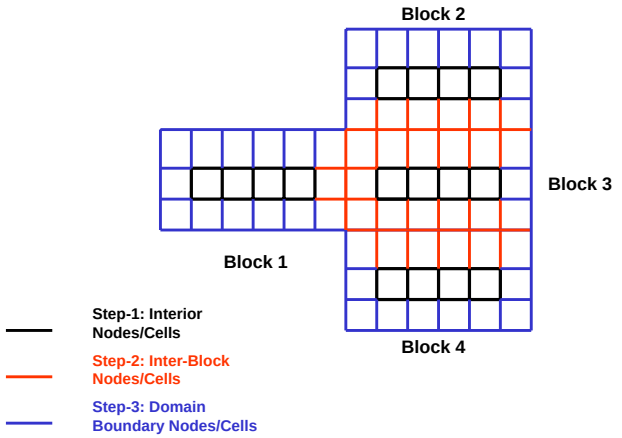
Data Structures to Aid in Multi-Block Discretization and Parallel Communication

- **Different strategies may be used to create the stencil at the inter-block boundary**
  - Since the block neighbor and boundary condition information is known for each side of every block, we can quickly determine the adjacent block processor numbers
- **Method-1: Information may be passed on the fly during the construction of the stencil at the edge**
  - Tends to be inefficient due to a large number of small-size packets of information exchanged
- **Method-2: Stencil may be constructed via an "accumulation" of forward and backward stencils at the edges (lecture 5)**
  - Sum the contributions from different processors at the edges at the end of stencil construction (cell-vertex schemes)
  - Requires additional logic to handle various stencil operators toward edges
- **Method-3: Information may be passed at the end of each iteration into buffers (halo or ghost cells/nodes) that "extend" each block**
  - Requires additional storage but is efficient
  - Ghost cells/nodes may also be used to perform boundary conditions

Method-1: "On the Fly" Stencil Construction

- **Pseudo-Code for multi-block, structured-grid simulations using method-1:**
  - Loop over blocks
    - Loop to construct discretization of stencil at internal nodes/cells
    - Loop to construct discretization of stencil at internal north boundary nodes/cells (reach across to north neighbor)
    - Loop to construct discretization of stencil at internal south boundary nodes/cells (reach across to south neighbor)
    - Loop to construct discretization of stencil at internal west boundary nodes/cells (reach across to west neighbor)
    - Loop to construct discretization of stencil at internal east boundary nodes/cells (reach across to east neighbor)
    - Loop to construct discretization of stencil at internal hither boundary nodes/cells (reach across to hither neighbor)
    - Loop to construct discretization of stencil at internal yonder boundary nodes/cells (reach across to yonder neighbor)
    - Construct discretization of stencil at block corner nodes/cells (reach across to two appropriate neighbors for each)
    - Perform boundary conditions on all non-inter-block faces
  - Update variables and make sure inter-block boundary nodes/cells are consistent

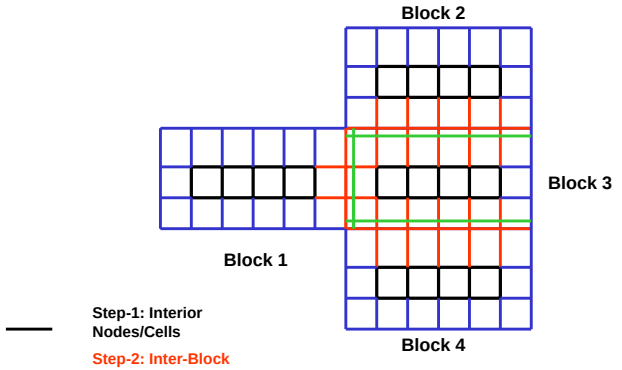
Example: "On the Fly" Stencil Construction



Method-2: "Accumulation" Stencil Construction

- **Pseudo-Code for multi-block, structured-grid simulations using method-2:**
  - Loop over blocks
    - Loop to construct discretization of stencil at internal nodes/cells
    - Loop to construct discretization of stencil at north boundary using combination of central and one-sided treatments
    - Loop to construct discretization of stencil at south boundary using combination of central and one-sided treatments
    - Loop to construct discretization of stencil at west boundary using combination of central and one-sided treatments
    - Loop to construct discretization of stencil at east boundary using combination of central and one-sided treatments
    - Loop to construct discretization of stencil at hither boundary using combination of central and one-sided treatments
    - Loop to construct discretization of stencil at yonder boundary using combination of central and one-sided treatments
    - Perform boundary conditions on all non-inter-block faces
  - Accumulate contributions at inter-block faces
    - Message pass (send/receive) boundary contributions
  - Update variables

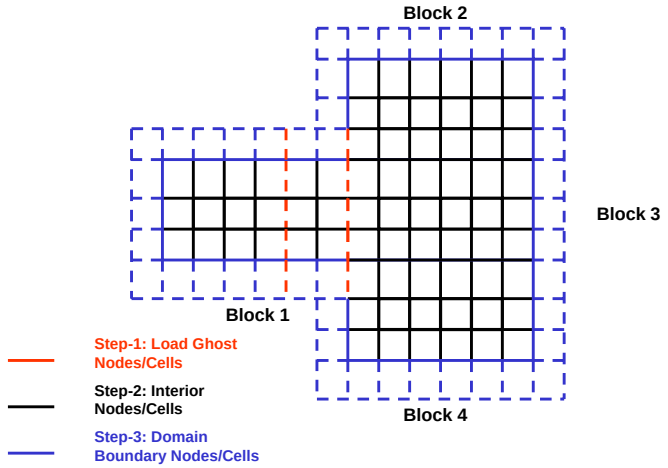
Example: "Accumulation" Stencil Construction



Method-3: "Ghost Cell/Node" Stencil Construction

- **Pseudo-Code for multi-block, structured-grid simulations using method-3:**
  - `Create enlarged-block data-structure that has additional planes beyond each face (halo or ghost nodes/cells)`
    - `Number of additional planes beyond each face is (stencil_size-1)/2`
  - `Read original blocks into enlarged-block data structure`
  - `Loop over enlarged-blocks`
    - `At inter-block boundaries, pack halo/ghost nodes/cell information with appropriate neighbor information`
    - `At domain boundaries, create appropriate halo/ghost node/cell information for boundary condition treatment`
  - `Loop over enlarged-blocks`
    - `Loop to construct discretization of stencil at all original block nodes/cells`
    - `Perform boundary conditions on all non-inter-block faces`
  - `Update variables`

Example: "Ghost Cell/Node" Stencil Construction



Multi-Block Unstructured-Grid Treatments

- **The multi-block data structure and inter-block stencil construction for unstructured-grids is similar**
  - Methods 1-3 can be constructed for unstructured-grids
  - Except instead of looping over faces/corners, the boundary is treated as an unstructured column vector of cells/nodes
  - The indirect-addressing associated with the randomness of the grid cells/nodes makes the treatments more tedious, time-consuming, and memory-consuming
  - For unstructured-grids, spatial/grid decomposition is often combined with processor/parallel decomposition
    - Unstructured grid-generation tools often create the grid over the entire domain as a single block

Project-2: Single-Block Decomposition

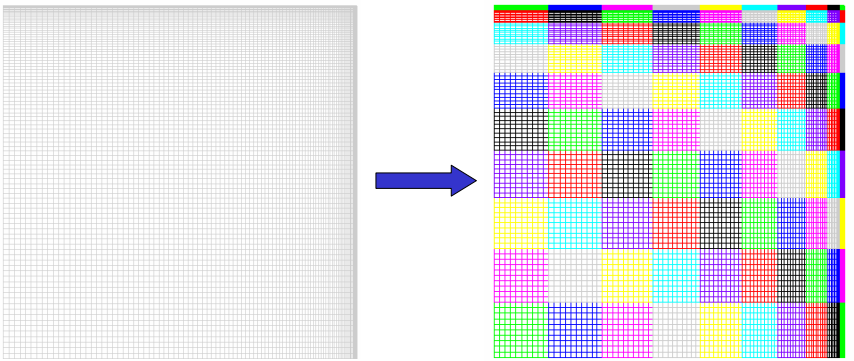
- **Now that we have shown that we can develop a single-block engineering simulation code, let's break up the problem into multiple blocks.**
- **For your simulation code developed under project-1, you need to write a code that will break up the domain into multiple blocks that will still run on a single processor**
- **This is an intermediate step prior to developing a code for multiple processors**

Project-2: Single-Block Decomposition

- **For the Heat Conduction (Default) problem, develop a code that will divide the domain into N x M blocks, each with dimensions of 1+(IMAX-1)/N x 1+(JMAX-1)/M**
  - Where IMAX = JMAX = 101 and 501 for default heat-conduction problem
  - The code should be completely general so that N and M can be arbitrary
- **Develop a multi-block data structure that writes out a connectivity file with the following information for each block:**
  - Global block number of each block
  - Neighbor block numbers of each block
  - Boundary conditions for each side of block
    - Could be unique to each side
    - Could have regions for each side
      - Number of sub-regions for each side (if you decide to have sub-regions)
      - Dimensions of each sub-region
      - Boundary condition for each sub-region
    - Could be defined point-by-point
  - Block orientation (all blocks will probably have the same orientation for our problems).
- **And multi-block grid and temperature files (Plot3D or other format) that has the coordinates and initial temperature along with**
  - Number of global blocks
  - Block dimensions of each global block

Project-2: Single-Block Decomposition

- **You should end up with something similar to:**



in the x', y' frame with associated connectivity and boundary condition file(s)

- **Project-3 will be to write a multi-block serial code that will solve your simulation problem**

Project-2: Single-Block Decomposition

- **Due Monday 10/28:**
  - An overview of your simulation problem
    - Describe the problem, algorithm, and boundary conditions
  - Listing of your spatial/grid decomposition code
  - A plot of your decomposed computational grid (or data decomposition) for two-different decompositions:
    - 10 x 10 (N=10, M=10) grid for heat-conduction problem
    - 5 x 4 (N=5, M=4) grid for heat-conduction problem
    - Using the 101 x 101 and 501 x 501 grids
  - A sample listing of your connectivity/boundary condition file(s)