Lecture 18 – Data Structures for Decomposed Domains on Parallel Computers

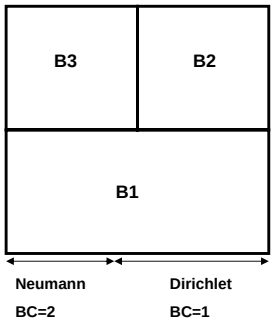
- **Now that we have discussed techniques for performing processor decomposition, we should also discuss the data structures that must be put in place to keep track of other block/processor boundaries**
- **This information can be kept as part of the "connectivity" file**

Additional Connectivity Information

- **In project 2, you developed a data structure to keep track of block neighbors for your multi-block solvers.**
- **This connectivity information kept track of**
  - Neighbor information on sides (and corners) of each block
  - Boundary condition options on physical domain boundaries
- **All blocks were kept in a single processor (project 3)**
- **All block numbers were considered to be in global space (ie they were all on the same processor)**
- **All blocks were assumed to have only a single neighbor or boundary condition to each side (or corner)**
- **All blocks had same orientation**
- **Now let's consider a more general capability**

Additional Connectivity Information

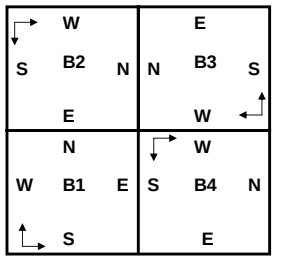
- **Additional connectivity information would be required if blocks could have more than one neighbor or boundary condition type per edge/corner**
- ***Sub-faces* can be used to enable multiple boundary conditions per side/corner**



The same idea also holds for processor boundaries

Additional Connectivity Information

- **Orientation of blocks can be non-uniform with any face matched to any other face of a neighbor**
- **Block neighbor orientations can also be kept as part of a connectivity file**
- **For instance:**
  - Let +i-direction=1, +j-direction=2
  - Block 1's orientation could be 1 2
    - (ie its i-direction is in the +i-direction and
    - it's j-direction is in the +j-direction
  - Block 2's orientation could be -2 1
    - (ie it's i-direction is in the -j-direction and
    - it's j-direction is in the +i-direction



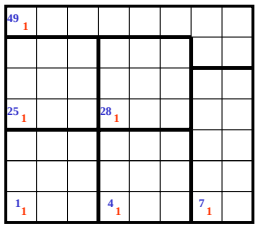
Another technique would be to just describe the neighboring face number for each block face

Virtual (or Local) Space

- **When blocks are partitioned to processors, virtual (local) block numbers must be maintained in addition with global block numbers**
  - For any given processor, the blocks will be looped from 1 to the number of blocks on that processor (ie virtual (local) space)
- **A mapping scheme must be created that can be used to determine the global block number for each virtual (local) block on a given processor**

Example of Virtual Block Information

- **Consider this example of an 8x7 blocks divided up into 6 processors. The PLB would be 56/6 or approximately 9 blocks per processor…a difficult decomposition.**
- **Multi-block grid, temperature, flow, etc files will exist for each processor**
- **A single global connectivity file could be created, or a virtual (local) connectivity file for each processor could be used**
- **Mappings between local and global block numbers should be created**
- **Each block must now keep track of its neighbor block numbers (local and global) and processor number**



Virtual block numbers

global block numbers

For each processor, blocks must be looped over local block numbers

Example of General Connectivity Files

- **For *general* configurations on parallel processors, a connectivity file could keep track of:**
  - Global block number, local block number, processor number
  - Orientation, OR
  - Number of south (face-1) sub-faces
  - Start index of south sub-face, end index of south sub-face, neighbor global block number, neighbor local block number, neighbor processor number, neighbor face number, neighbor sub-face start index, neighbor sub-face end index
  - Number of north (face-2) sub-faces
  - Start index of north sub-face, end index of north sub-face, neighbor global block number, neighbor local block number, neighbor processor number, neighbor face number, neighbor sub-face start index, neighbor sub-face end index
  - Etc.

Project 4

- **Take the decomposed set of blocks and connectivity file for the 10x10 and 5x4 block decompositions that you created under project-2, and write a code that will map them onto P processors.**
  - Use what information you know about the grid to maximize data-locality and minimize communication costs
  - You can write your own code or use existing libraries such as Metis
    - I want you to write your own code, however.
  - In your new connectivity file(s), somehow mark the neighbor information to keep track of neighbor processor numbers and a way to map local (virtual) block numbers to global block numbers

Project 4

- Write out either a single new connectivity file or P connectivity files, multi-block grid, and multi-block initial temperature files with names corresponding to rank number (eg conn.dat.p0, xyz.dat.p0, temp.dat.p0) so that each processor can "read-and-go" in project 5.
  - I have put an example how to write sequential files in "Additional Material" on smartsite
- You can combine this code with your project-2 spatial decomposition code if you wish.
  - That way, you can decompose and map in one step.
- Or you can create a new code just for the parallel/processor mapping.

- **The goal is for each processor in the project 5 code to read only it's information and start iterating! No processor should contain the global information (except possibly upon convergence to agglomerate the blocks back to the global domain for plotting purposes)**

Project 4

- **Due Wednesday, November 20th**
  - Brief description of your project
  - How you decomposed the domain in project 2
  - Methods used to map the decomposed grid onto P processors
  - Listing of parallel/processor decomposition code
  - Demonstration that your code works for 4 and 6 processors on the 10x10 and 5x4 set of blocks
    - Write out the connectivity file(s) for the 4 and 6 processor parallel/processor decompositions

Parallel Performance Analysis

- **Objectives of Parallel Computing**
  - Ultimately, in parallel computing, we intend to achieve:
    - Faster execution speed
      - Enable multiple analysis in a fixed amount of time
      - Decrease time necessary to complete one solution
      - Increase the level of modeling of our physical system
      - Enable paradigm shifts in the way that computational science is used.
    - Lower cost
      - Strictly speaking, it is nearly impossible to obtain lower cost when using a parallel computer (parallel processing overhead, additional expense of interconnection network, etc.)
      - Lower cost can be derived from additional benefits that result from the ability to execute a given program in a shorter amount of time (ie lower labor costs)
      - However, we must strive to maintain the cost of parallel computing from departing severely from single processor computations

Objectives of Parallel Computing

- **We would like to achieve these goals using *reasonable* resources:**
  - High parallel efficiency
  - Large computational speedups / scalability
  - Low development cost / investment
  - Efficient memory scalability

Objectives of Parallel Computing

- **In order to achieve these goals we need to understand:**
  - Single-processor performance models and efficient programming techniques
  - Parallel processing performance and bottlenecks
  - Mapping between a specific algorithm and the computational hardware available
- **We have discussed all of these somewhat already. Let's look at these further.**

Objectives of Parallel Computing

- **In summary, our objective is to learn about the necessary models to:**

⇨*Guide algorithmic software development by using predictive models of performance*

⇨ *Determine, a priori, the best choice of algorithm to be executed on a parallel computer*

Example

- **Building of the Hadrian's wall in the border between England and Scotland. Commissioned by Emperor Hadrian of Rome in 122 A.D. (73.3 miles in length).**
- **Perfect example of a parallel computing problem.**
- **Large size, embarrassingly parallel, parallel I/O**

Example

- **A large number of legionnaires (processors) can be used at the same time with high efficiency.**
- **Global communication is not required: each legionnaire only needs to talk to the two legionnaires to his right and left.**
- **SPMD (Single Program Multiple Data) strategy is being used: legionnaires are replicated along the wall.**
- **Each legionnaire can retrieve his own mortar and stone (parallel I/O) assuming there is no bottleneck at the supply location.**

Example

- **Consider instead the building of the tower of Babel. Is there any parallelism to be found in this problem? Some, at a floor level, but this problem is inherently *sequential.* Similar observations can be made in the time-accurate integration of PDEs unless you can think cleverly about the problem and reduce it.**

- **Your job as a parallel scientist is to rediscover the way of building a tower such that additional areas of parallelism can be found and exploited.**

Single-Processor Performance Factors

- **Factors affecting single-processor performance include:**
  - Mflop rating
  - Vector vs. Cache architectures
  - Memory hierarchy
  - Microprocessor architectural issues (no. of ops/cycle, no. of load/store ops/cycle, pipelining ability, etc.)
  - Careful decomposition and programming issues

Multiple-Processor Performance Factors

- **Things affecting multiple-processor performance in parallel systems include:**
  - CPU scalability
  - Memory scalability
  - Interconnection network
  - Bandwidth and latency issues
  - Problem size and granularity
  - *How many processors can we use efficiently?*