Goal
Implement and evaluate a parallel 2D Heat Equation solver using MPI.

Details
In this assignment, we will re-implement the 2D Heat Equation Solver from Lab Assignment 3. However, instead of parallelizing using OpenMP, we will rely on MPI. For the most part, the parallelization approach is similar in that the 2D domain of points is decomposed over the available cores, but now the data partitions and the cores are spread across a distributed memory system.

To parallelize the solver with MPI we will implement a 4-way (2x2) domain composition and a 9-way (3x3) domain decomposition, allowing us to parallelize the 2D solver across 4 processors and 9 processors respectively.

With OpenMP, decomposition of the loop iterations across the threads is mostly hidden from the programmer. However, with MPI the programmer is directly responsible for how the input data is partitioned across the processors. For example, in the 4 processor decomposition above, MPI Rank 0 can be instructed to compute only the values in the lower left region of the input array.

Note that for data points that are on the partition boundaries, your program must implement some mechanism like halos (ghost cells) to work correctly. Because each calculation requires information about neighboring data points, data point calculations along the partition boundaries may require data points from other partitions. After each iteration information about these data points from other partitions must be exchanged between the separate processors.

As we compute more iterations, the data points will eventually converge, in that they vary only slightly between during subsequent iterations. For example, if the boundary values are all initialized to 0, eventually the entire input will converge to 0, independent of the initialization of the inner data points. Depending on the initialization values, this convergence can take a variable amount of iterations. However, if we can check for convergence, we can terminate the computation before computing the specified number of iterations.

To test for convergence, for all data points calculate the difference in value between current iteration and previous iteration. If any difference is above a certain threshold, assume solver has not yet converged.
Steps

1. Starting with your code from Lab Assignment 3, re-write to partition data across 4 processors using the 2x2 grid decomposition. Requests for a different number of MPI Ranks can return an error. Begin your implementation using a small input size (100x100) and a small number of iterations (10).

2. Expand the 4-processor implementation to exchange halo data or ghost cells between iterations using MPI point-to-point communications.

3. Expand the 4-processor implementation to test for convergence using an MPI collective communication, breaking the loop early if the convergence criteria is met. Use a threshold of 0.001 for the convergence test.

4. Test the 4-processor implementation on Cerberus, using slurm to schedule the MPI-processes across the NUC cluster. See Lab 1 and Lab 6 for examples using slurm. To verify the correctness of your implementation, you can initialize the input such that one partition contains high values while other partitions contain low values, and observe as the high values are propagated across the partitions.

5. Expand the 4-processor implementation to handle 9 processes. You can include this version in a separate source file. You can verify the correctness of this implementation using a similar strategy as the 4-processor approach.

6. Perform an experiment similar to that of Lab Assignment 3. Test the serial, 4-processor, and 9-processor solvers with input sizes of 100x100, 1000x1000, and 5000x5000, and 1000 iterations.

Submission

To submit this assignment, create the following directory in your git repository:

`lastname_parall/elab_assignment_3`

Within this directory, please include the following:

1. The source code of your C++ MPI 2D Heat Equation Solvers.

2. A .pdf or .txt document containing a summary of your observations and results. Include a figure or table summarizing the results of the experiment in Step 6. For example, your table could resemble the following, with the empty cells replaced with the appropriate run-times, where \( p \) represents the number of processors, and \( n \) represents the input size (\( n \times n \)).

<table>
<thead>
<tr>
<th></th>
<th>( p = 1 )</th>
<th>( p = 4 ) (2x2)</th>
<th>( p = 9 ) (3x3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n=100 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n=1000 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n=5000 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assigned: February 23, 2018
Last Update: February 23, 2018
Due: March 2, 2018