Benjamin Kaiser
11-23-2016
CSC 410 Parallel Programming
Dr. Christer Karlsson

The following is a solution using Foster's design methodology.

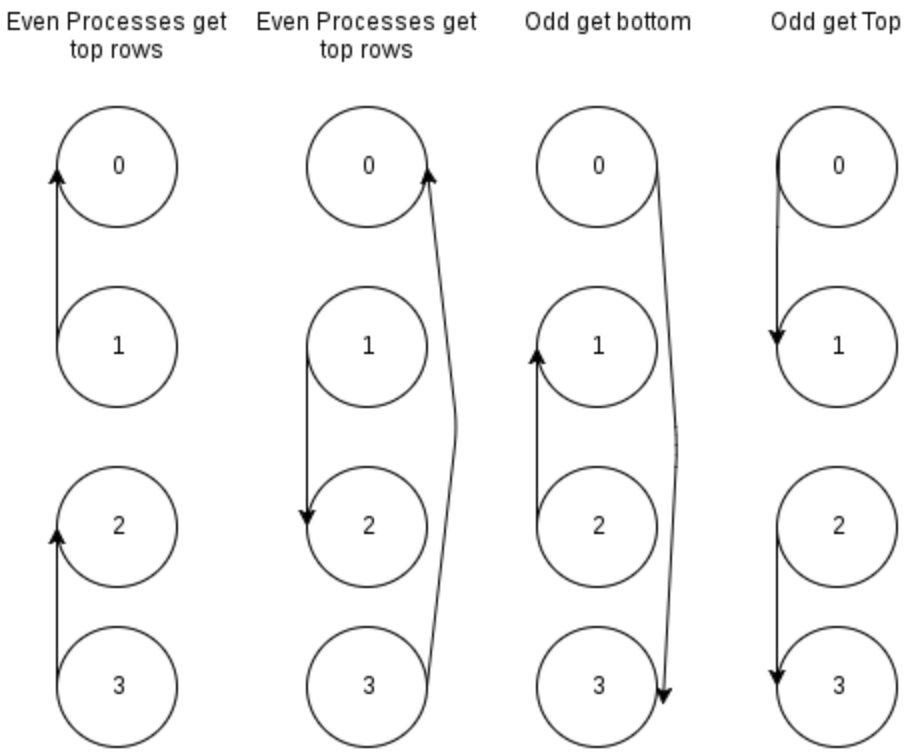
Partitioning:

This is the key to the entire program. If we don't know how to partition, then we don't know how to communicate and if we don't know how to communicate then we can't agglomerate/map. If we partition across the rows, then an entire row is sent to each process. However we must realize that if we partition via rows, we will be limited on the size of matrix that is given us. For example if we have a 3 x X matrix, we are limited to only using max of 3 processors. If the X variable is big, then this could result in a not very big speedup. We could also split out by columns which would have the same issue as the rows - it would not be very scalable. This would have the added annoyance of trying to figure out how to print. Another thing we could do is split the array into square or near to square blocks of the array. This would be the hardest to figure out printing but it would also be the most scalable.

For ease and an introduction into MPI, I am going to go with the row partitioning idea. This means that each process gets a row or groups of rows.

Communication:

All printing will be done from process 0. I feel this will be the easiest. I could have each process print it's own stuff but that's just dumb and non-deterministic. An array of alive points will be passed to each process which will then determine if it controls those points. As each item in the grid array is calculated, the nodes communicate with each other and the state of the item(s) are sent back and used to calculate.



Agglomeration:

This again is that each row of the data grid array will be processed by an individual process which will then iterate through the array and calculate. Each process will hold the data that it receives from the other processes above and below it in it's own individual arrays. This

Mapping:

The mapping will be in a cyclic fashion as individual rows. The first row will go to the first process the second and so on. When the program runs out of processes, the next row goes back to the first process and so on and so forth.