Red-Black Multigrid Optimizations of an Ising Percolation Model

Parker Van Roy

Boston University College of Engineering
ENG EC 526 - Prof. Brower

1. Introduction

This project concerns the adaptation of one file, phi.cpp, which is a simulated 2-D grid that solves randomly generated periodic Ising bond grids where the randomization function has been pre-set to generate ‘middle ground’ clusters that produce large, snaky clusters while not connecting the majority of the grid.

The function that was the focus of this project has a few parameters:
‘Label’, A 2-D array of integer labels. These labels were given a positive or negative value to indicate the magnetic direction. The number of the label indicated the position on the grid, and as clusters were identified the label of all elements would change to the lowest number found to give each cluster its own number.
‘Bond’, A 2-D array of 4-element arrays of bools, indicating whether there was a bond in each direction.
‘L’, the x & y lengths of the maximum grid
‘S’, the x & y lengths of the current grid (new in multigrid implementation)

The implementation of a recursive multigrid model to replace an iterative model was the focus of this project.

Since the iterative model given would calculate new labels for the entire grid before updating, an attempt was made to keep optimizations in line with this philosophy.

OpenACC was used to parallelize this and any helper functions. Everything within the function call was processed on GPU with OpenACC.

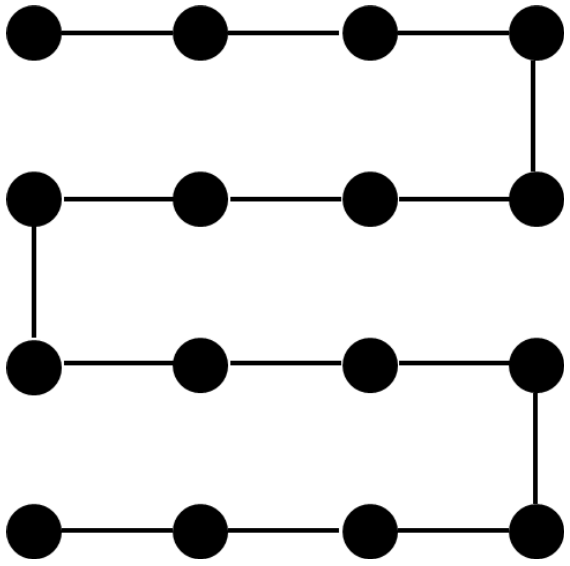
2. Set-up of a Recursive Multigrid and Preliminary Results

To begin on the project, a recursive multigrid system was implemented to replace a direct iterative solution. The multigrid is a v-cycle as recommended by Brower [1].

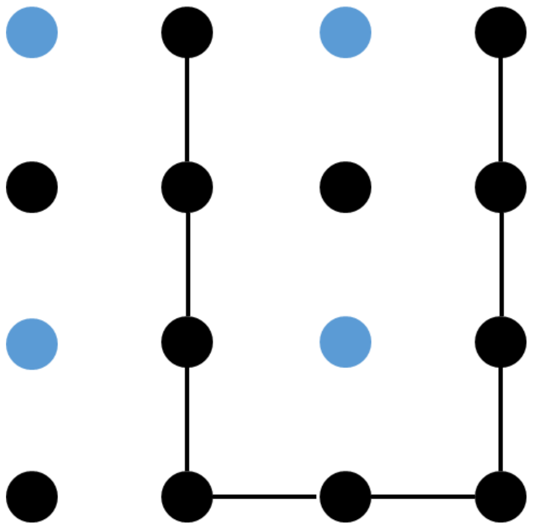
The cycle begins at the maximum size of the grid, sends a recursive call with a s/2 sized coarse grid, replaces values that were changed by the coarse grid, and iterates. The grid does not call multigrid at s==2 for obvious reasons.

The multi-grid function was not more efficient for values of L up to 256. (The provided file is non-functional for larger L unfortunately and I haven’t been fix the bug yet)

The serial iteration must iterate a maximum of L-1 times to ensure full cluster detection. This is illustrated below. This number can be found by calculating (L-1)*L horizontal bonds + L-1 vertical bonds. The number of necessary actions on the L^2 grid is thus approximated to L^2* L^2 = L^4. On N = 256, this number is 2^32 actions.



Given a grid (pictured below, blue indicates points on coarse grid) where bonds are only on non-coarsened points, basic multigrid efficiency gains look problematic. The maximum number of iterations is decreased to (L-1)*L/2 vertical bonds + L/2 horizontal bonds, which approximates to L^4/2 actions, but this is at the cost of calculating and copying L^2/4 points * 4 directions which is in effect SLOWER! A simpler way to think about this is in creating a new coarse grid for each of the 4 points in a 4*4 grid, it takes 2 iterations to connect every grid when returning to fine but each point but an L/4 iteration still needs to be done for each of L^2 point, and a log2(L) needs to be multiplied for the total number of actions due to the recursive function. This results in O(L^3*log2(L)). As the scale up to 256 this simply does not provide enough efficiency, and I am not sure if this function performs better at any scale.



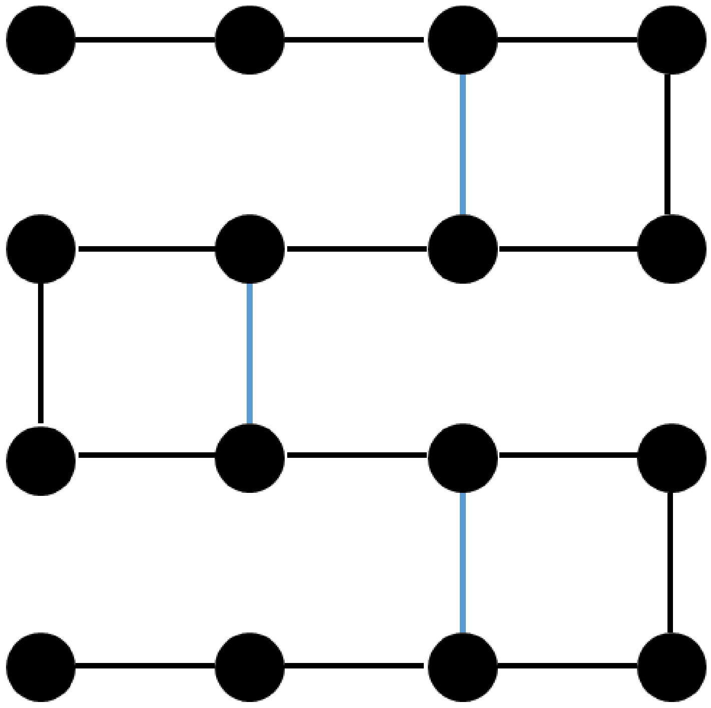
This is, of course, just a start in analyzing a multigrid function, and only uses straight double bonds, but path-finding to find any path to a two-distance point is clearly not efficient.

3. Extraneous Red-Black Optimizations

Two small changes were made to the original phi function before designing a multigrid function that was more efficient that the serial function.

1) Closing boxes in original grid:

The most significant change came from a ‘square closing’ function used in the lattice percolation to reduce iterations. By iterating over the grid and testing a 2*2 box for the 4 bonds in it, it takes 4*L^2 actions to close any possible boxes in a grid. Iterating this the maximum number of times to close boxes formed by other boxes is L^2 times, but a simple position update on box close reduces the max by ½ and makes the max iterations equal to the number of box closings. Using one iteration of this reduces a maximal path by a significant amount- L^2-1 iterations from beginning to end to (L-2)*L + L-1 = L^2-L-1 iterations. Since the bond function is parallelizable, this is great! By iterating the bond function as many times as possible (Max L^2/2), the max iterations is now (L-1 + L-1) on a fully snaky grid! The time with the bond mechanism is thus O(L^2/2) amortized.

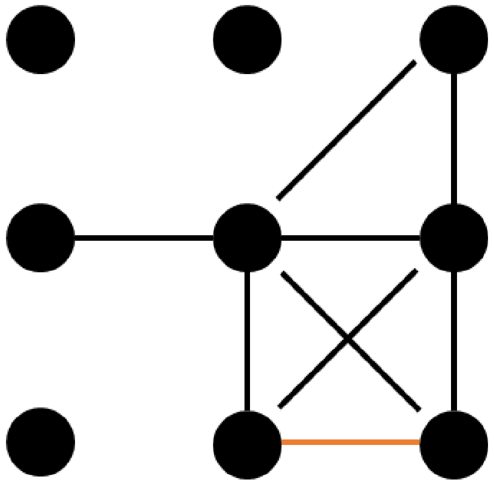


2) Red-Black iterations:

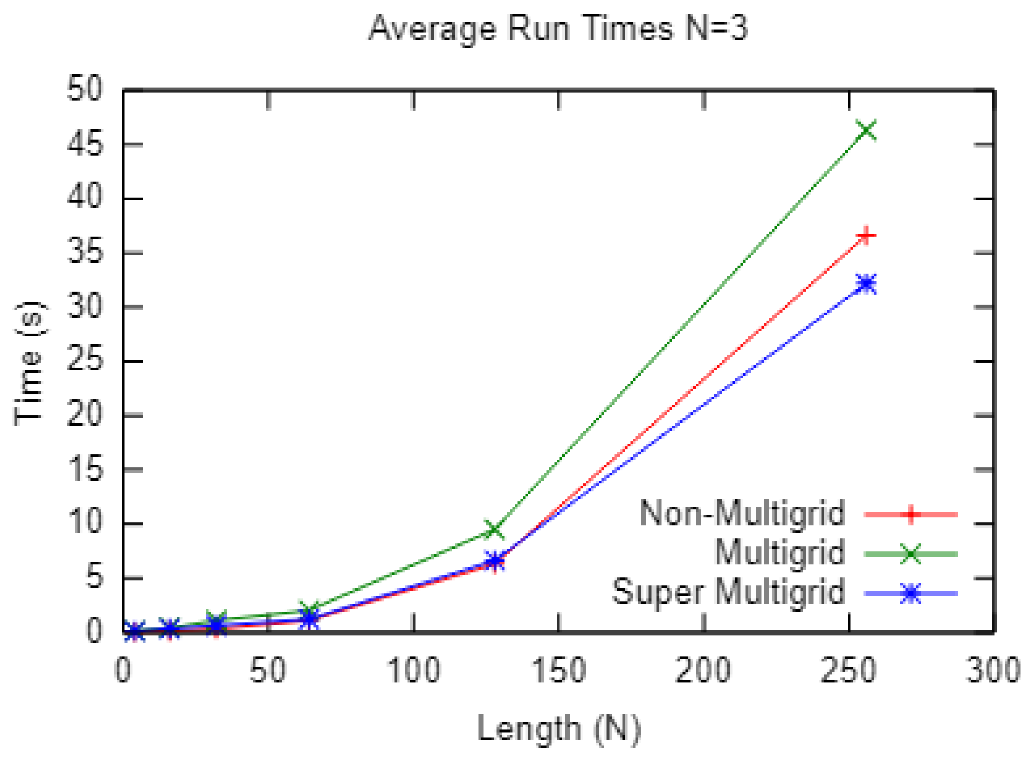
Simply, by iterating the labeling of the grid in a red-black manner alternating points on the original grid instead of a copy, grid bonding can be 'doubled' in speed with reduced copying! (Since it was parallelized already this isn't really significant though)

4. Apply Red-Black Optimization to Recursive Multigrid and Results

After examining the way in which the square-closing increased speed, it was implemented into each layer's recursive calls as a method of increasing speed. To reduce writing and include square closing as a fundamental method, the bond array was changed from representing up/down/left/right to down/right/down+right/up+right. In this way, bonds are one directional and bonds to adjacent points may need to be called from the adjacent point. In including this half-power distance as a fundamental data type, by using the half power and a double bond a cage can be created around each point so that path finding is unnecessary. The inclusion also means that some iteration time is move from iterating through bonds to creating the data structure, but the time used is the same. This method also increases square closing by only checking if the diagonal bonds cross, which is slightly faster!



By iterating between diagonal bonds and square closing, it is possible to close any box size on the grid. While iterating this way is slow, by using double bonds this can be done very efficiently in recursion! Since iteration of percolation is done upwards only, there is no issue with creating bonds effectively on the down cycle. The total time of this type of system, with the knowledge that 2 iterations are needed at each stage, is L^2 log2(L) without considering the diagonal bonds and box-closing iterations. As these iterations each occur in parallel however, the number of iterations depending on the size of the grid is more of an issue of number of operations per object, as closing squares is 8 operations each point and box closing is 6 per box closed. Since this number of operations/if statements/etc. is much greater than before to generate this data structure, it makes sense that the efficiencies of this method only show when the size of the grid is large enough. As shown by the run times, this seems to be the case as the multigrid method becomes more efficient at L=256!



5. Further Thoughts on Optimization

The functions written to analyze this model were NOT DYNAMIC i.e. do not examine the probability function and descent of the multigrid. One way I am very interested in pursuing is using the bounding distance of 2 with double bonds and ½ powers to coarsen the grid only at points where there is a bonded edge. This is in effect using a cluster finding algorithm to find boxes to close and fill as opposed to boxes to fill from one end to the other. By enclosing a large area with a label it is very efficient to percolate it through an area and by storing the information in some form of dynamic memory, it is possible to identify clusters before filling details better than the straight lines of multigrid.

Another strategy would be to ‘abuse’ the probability function to determine likelihood of bonds/boxes/other patterns at each level of the multigrid so there is a clear point of what operation to do as well as how many iterations is efficient. For example, as more boxes are completed there is a lower likelihood that each iteration will create a new box, and completing boxes is only efficient if there are enough of them to do on the grid, otherwise 2 normal percolation operations suffices. However, since recursion depends on completing boxes, it also must depend on whether the coarser grid of recursion is likely to be valuable or not and with or without completed boxes. (‘Cheap’ double bonds of straight lines without completing boxes are

only efficient if there is are 4 bonds in a row connecting coarsened points- otherwise, it is better to just iterate twice!)
A final strategy conception goes deeper into the idea of red-black by instead making two recursive calls and using top-left and bottom-right of each 2*2 to form double bonds, allowing for less iterations but more copying and recursion.
It may also be of use to calculate indices rather than copy indices, as the majority of the copying is the labels (int) as opposed to the bonds (4*bool) and labels are unchanged. However, this makes code much less readable as well.