Parallel Multigrid Percolation Cluster Detection

Parker Van Roy

EC526
Brower

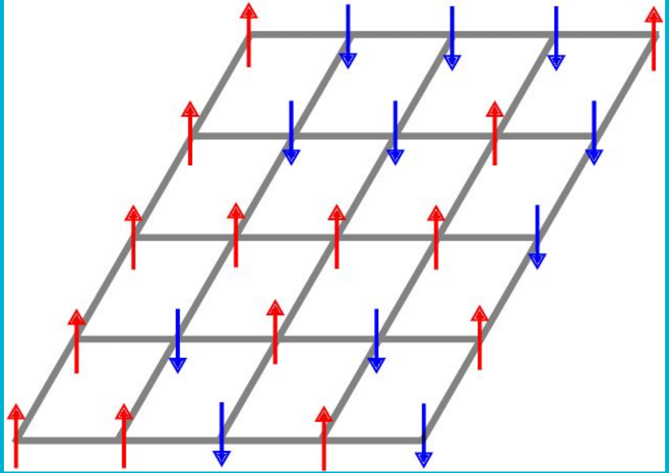
Ising Percolation Cluster Problem

Magnetic polarities tend to align.

We can count the energy of a plane of magnetic poles where we sum for each bonded/neighboring pair of poles -1 if they are the same and +1 if they are different by creating this as an undirected graph

We can relax this grid with the Swendsen-Wang algorithm by ‘percolating’ random bonds then randomly flipping where these bonded poles match

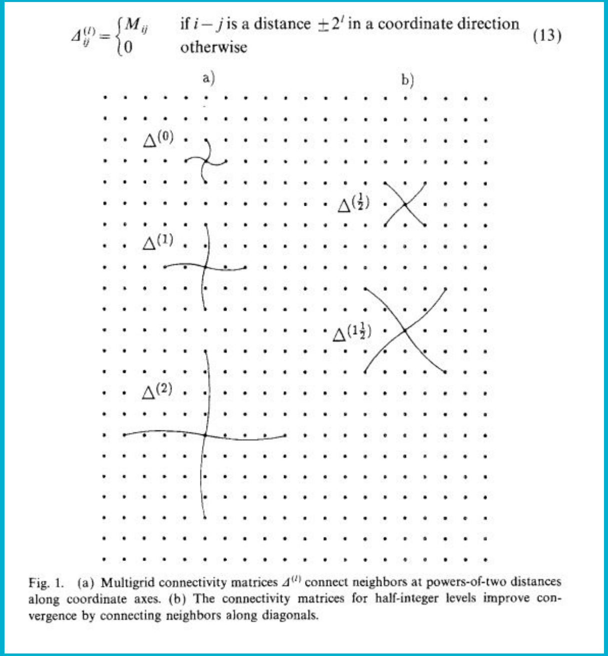
From a point we can serially find a cluster of like poles by BFS or DFS



Multigrid Algorithm

We are looking rather at relaxing based on a fixed point. We can derive the max distance of points in its cluster.

With a single V cycle of multigrid fine-coarse-fine we can distinguish where prior separate clusters can be combined by expanding a connectivity matrix by powers of two each iteration.

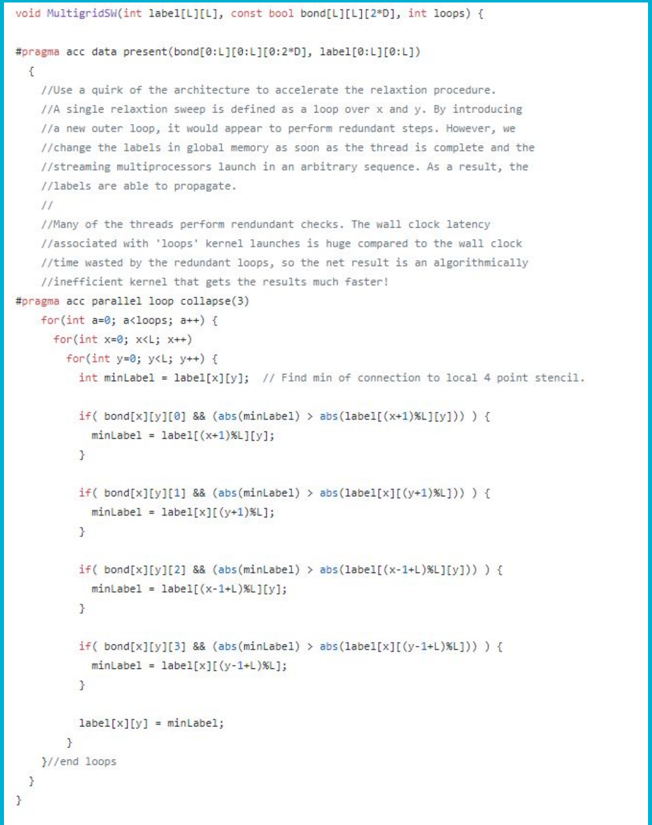


$$\Delta_{ij}^{(l)} = \begin{cases} M_{ij} & \text{if } i-j \text{ is a distance } \pm 2^l \text{ in a coordinate direction} \\ 0 & \text{otherwise} \end{cases} \qquad (13)$$

Fig. 1. (a) Multigrid connectivity matrices $\Delta^{(l)}$ connect neighbors at powers-of-two distances along coordinate axes. (b) The connectivity matrices for half-integer levels improve convergence by connecting neighbors along diagonals.

TODO

The goal is to adapt the connection machine code to an efficient openacc form so that major speed increases can be seen in order to be applied to a variety of algorithms.



```
void MultigridSW(int label[L][L], const bool bond[L][L][2*D], int loops) {

#pragma acc data present(bond[0:L][0:L][0:2*D], label[0:L][0:L])
  {
    //Use a quirk of the architecture to accelerate the relaxtion procedure.
    //A single relaxtion sweep is defined as a loop over x and y. By introducing
    //a new outer loop, it would appear to perform redundant steps. However, we
    //change the labels in global memory as soon as the thread is complete and the
    //streaming multiprocessors launch in an arbitrary sequence. As a result, the
    //labels are able to propagate.
    //
    //Many of the threads perform rendundant checks. The wall clock latency
    //associated with 'loops' kernel launches is huge compared to the wall clock
    //time wasted by the redundant loops, so the net result is an algorithmically
    //inefficient kernel that gets the results much faster!
#pragma acc parallel loop collapse(3)
    for(int a=0; a<loops; a++) {
      for(int x=0; x<L; x++)
        for(int y=0; y<L; y++) {
          int minLabel = label[x][y];  // Find min of connection to local 4 point stencil.

          if( bond[x][y][0] && (abs(minLabel) > abs(label[(x+1)%L][y])) ) {
            minLabel = label[(x+1)%L][y];
          }

          if( bond[x][y][1] && (abs(minLabel) > abs(label[x][(y+1)%L])) ) {
            minLabel = label[x][(y+1)%L];
          }

          if( bond[x][y][2] && (abs(minLabel) > abs(label[(x-1+L)%L][y])) ) {
            minLabel = label[(x-1+L)%L][y];
          }

          if( bond[x][y][3] && (abs(minLabel) > abs(label[x][(y-1+L)%L])) ) {
            minLabel = label[x][(y-1+L)%L];
          }

          label[x][y] = minLabel;
        }
    }//end loops
  }
}
```