CS 320/CSE 302/ECE 392

Introduction to Parallel Programming for Scientists and Engineers
MP4 - Due Friday, April 10

The objective of this MP is to write two different OpenMP parallel programs to iteratively solve a two-dimensional LaPlace equation. A possible serial version of the code is (taken from *In search of clusters* by G. Puffiest. Prentice Hall, 1997):

```
do forever
     max_change = 0  ! the largest error we've seen. Initially 0.
     do i=2,N-1      ! do all rows, but no outer boundary rows
          do j=2,N-1 ! do all elements in a row, but not boundaries.
               old_value=v(i,j)! record v(i,j) before changing it

               ! replace each value with the average of its neighbors
               v(i,j)=(v(i-1,j)+v(i+1,j)+v(i,j-1)+v(i,j+1))/4

               ! keep max_change at the largest magnitude change we've seen
               max_change=max(max-change,abs(old_value - v(i,j))

          end do
     end do
     if max_change < close_enough then leave do forever
end do
```

Notice that the boundaries of the array v contain what is called the boundary conditions. Those are constant values at the boundary of the region where a differential equation is being solved.

Write a first parallel version following the wavefront method discussed in class. As usual, implement the vector operations using parallel DO loops.

The second parallel version will compute the values in an order different than the order of the original sequential version and, in fact, will contain race conditions. However, in many cases this form produces correct results and could be faster than the first parallel form. This second form will be implemented by parallelizing the `i` loop. A private variable `max_row_change` could be used to keep track of the maximum change to each row within the `j` loop. Then, just after the `j` loop and enclosed within a critical section, `max_change` should be updated. The termination condition is the same shown in the loop above.
The `i` loop could be stripmined to reduce the number of critical sections executed by each sweep of the `v` matrix. In this case `max_row_change` would keep track of the maximum change for a set of rows.

For the extra 50 points you are requested to implement a third version of the previous loop. It should be a blocked version of the wavefront method but implemented by parallelizing the outer loop. Synchronization operations should be used to enforce the wavefront ordering of execution. You should implement the synchronization primitives as busy waiting operations and use the OpenMP `FLUSH` primitive to guarantee correctness.

For this MP, set N=250. The boundary conditions should be `v(1,:)=1:250, v(:,1)=1:250, v(:,250)=250:1:-1,` and `v(250,:)=250:1:-1.`