Da Lu

Parallel HW3

Game of life

4/22/14

1 a)

The decomposition of our problem is based on data geometric decomposition. We divide the array (our array) into blocks. And each block will run on different processors. Each block has its own process and each block updates its own block. For our program, I divide the 16\*16 matrix by row based on the input. The decomposition is linear in row dimension.

b)

In our problem, if the two blocks are adjacent in the 16\*16 matrix, we only need to synchronize the values that are on the boundary. The boundary is just the first and last row for each block.

c)

My algorithm takes the input p, number of processors and divide 16\*16 matrix into p blocks by row. Then it creates p processes and o arrays for each block. During each of the 64 iterations, each blocks only updates its own block. However, we need to worry about the rows that are on the boundary for each block. For example, lets say we have 4 blocks A,B,C,D. A is adjacent to B,D. B is adjacent to A,C, and C is adjacent to B,D, D is adjacent to A,C. I divide the 4 blocks into two groups, <A,C>, <B,D>. A,C are not adjacent to each other so is B and D. For each iteration, my algorithm would do 2 steps. First step is to send information from A->B and A->D,C->D ,C->B. After this step is finished, my program send information from B->A and D->A, D->C, B->C. After the synchronization, data from all the blocks will be merged into a single array, and we are done.

2

a)

Advantages of this decomposition would be that it can give us n^2 blocks, while the horizontal slicing would give us most n blocks. Theoretically, the decomposition can run n^2 processes at the time and thus improve the overall parallelism. Disadvantages of this decomposition would be that each process will have more adjacent processes, while horizontal slicing only has 2 adjacent process, first and last row. So there will be more interference because processes need to communicate with each other more if we use this decomposition.

b)

new decomposition N by N:

Size of each MPI message: n/N

Number of MPI message: 8 \* N^2

Total size of MPI message: 8Nn

horizontal slicing N by N:

Size of each MIP message: n

Number of MPI message: 2 \* N ^2

Total size of MPI message: 2 \* n \* N^2

c)

new decomposition:

memory overhead at each process: 4 \* (n/N ) + 4

horizontal slicing:

memory overhead at each process: 2n

d)

new decomposition:

we see that total size of MPI message and total memory overhead are both

4nK + 4K^2 .

horizontal slicing:

we see that total size of MPI message and total memory overhead are both

2nK^2.

It is pretty clear to us that the horizontal slicing will have greater costs for MPI messages and memory overhead. However, the new decomposition costs more to communicate between processors. There, we have the trade off between communication frequency and load.

e)

Lets say we have a N\*N matrix, and each process Pi,j represents a block, i is the row index and j is the column index.

So there will be actually take only 4 steps. Lets first divide all the blocks into 2 groups A and B. All the blocks in A are not adjacent to each other and all the blocks in B are not adjacent to each other. Step 1, send all the information in group A to B, A->B. Step 2, send all the information in B->A. Then we divide blocks in-group A into A1 and A2 and divide blocks in group B into B1 and B2. Step3, send information A1->A2, B1->B2. Step 4, send information from A2->A1, B->B2. Below is a more detailed description.

1. for Pi,j, if i is even, then send its first row to Pi-1,j and last row to Pi+1,j
2. for Pi,j, if i is odd, then send its first row to Pi-1,j and last row to Pi+1,j
3. for Pi,j, if j is even, then send its first column to Pi,j-1 and last column to Pi,j+1
4. for Pi,j, if j is odd, then send its first column to Pi,j-1 and last column to Pi,j+1
5. for Pi,j, if i is even, then send its element in first row and first column to Pi-1,j-1 and the element in last row and last column to Pi+1,j+1
6. for Pi,j, if i is odd, then send its element in last row and first column to Pi+1,j-1 and the element in first row and last column to Pi-1,j+1
7. for Pi,j, if j is even, then send its element in first row and first column to Pi-1,j-1 and the element in last row and last column to Pi+1,j+1
8. for Pi,j, if j is odd, then send its element in last row and first column to Pi+1,j-1 and the element in first row and last column to Pi-1,j+1

So this algorithm will finish in 4 steps.