Una semplice struttura dati parallela (base per un Automa Cellulare)

Questa compito prevede l'implementazione di una semplice struttura dati parallela. La struttura è a due dimensioni, rappresentante una maglia regolare di punti (celle), suddiviso in fette (es: orizzontali, per iniziare), con ogni fetta assegnato ad un processo diverso. Nella forma più semplice, la struttura dei dati completa è (allocando staticamente):

```
double x[maxn][maxn];
```

e vogliamo fare in modo che ogni processore ha una parte locale:

```
double xlocal[maxn][maxn/size];
```

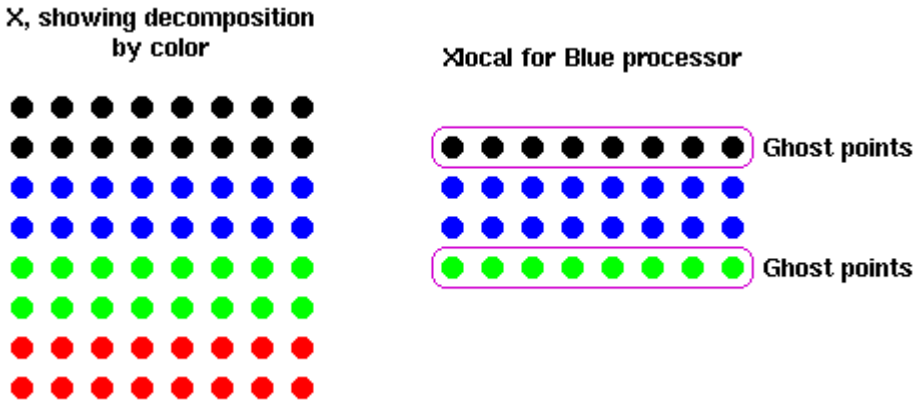
dove `size` dimensione è la dimensione del comunicatore (ad esempio, il numero di processori).

Per il calcolo che andremo a svolgere in questa struttura dati, abbiamo bisogno dei valori "adiacenti" di ogni processo. Quindi, supponiamo che per calcolare una nuova x [i] [j], abbiamo bisogno dei valori:

```
x[i][j+1]
x[i][j-1]
x[i+1][j]
x[i-1][j]
```

Accedere agli ultimi due valori potrebbe essere un problema se non sono presenti in `xlocal` ma invece su processori adiacenti. Per gestire questa difficoltà, è prassi comune definire i punti "ghost" che contengono i valori di questi punti adiacenti.

Scrivere un codice per copiare e dividere l'array `x` in strisce di uguali dimensioni e di copiare i bordi adiacenti ai processori vicini. Si supponga che `x` è `maxN` × `maxN`, e che `maxN` è equamente diviso per il numero di processori. Per semplicità, si può assumere una dimensione fissa per l'array e un valore fisso (o minimo) numero di processori.



Per testare la routine, ogni processore riempe la propria porzione con il rank del processo, e il ghostpoints con il valore -1. Dopo che lo scambio avviene, verificare che i ghostpoints hanno il giusto valore (i.e. non piu' -1 ma il rango del processore "vicino"). Si supponga che il dominio **non**

è toroidale, cioè il processo in alto (rank size = - 1) invia solo e riceve i dati da quello di cui esso (rank size = - 2) e il processo in basso (rank = 0) invia solo e riceve i dati da quello sopra di esso (rank = 1). Si consideri un maxN di 12 e usare 4 processori per iniziare.

Per questo esercizio, usare unicamente comunicazioni bloccanti, tramite MPI_Send e MPI_Recv, o al piu' MPI_SendRecv.

Modifica 1

Prevedere una struttura iterativa imbarazzantamente parallela. L'utente specifica il numero di passi, ed ogni passo il valore della cella è uguale a:

```
x[i][j] = f(x[i][j]) dove f è una funzione algebrica (somma,
prodotto, etc)
```

Modifica 2

Allocare dinamicamente i dati, utilizzando altresì una struttura di tipo Master-slave, dove c'e' un Master che gestisce l'intera matrice e ogni schiavo riceve la porzione in base al proprio rango…

Modifica 3

Usare routine non-blocking point-to-point invece delle routine bloccanti. Sostituire il MPI_Send e MPI_Recv routine con MPI_Isend e MPI_Irecv e utilizzare MPI_Wait o MPI_Waitall per verificare il completamento delle operazioni non bloccante.

Si consiglia di utilizzare queste routine MPI nella soluzione: MPI_Isend, MPI_Irecv, MPI_Waitall, etc.

Modifica 4

Utilizzare derived datatypes e topologie virtuali…