Message Passing Interface (MPI)

Διαχείριση Εικονικών Τοπολογιών

(συνέχεια)

MPI_CARTDIM_GET

- **Η συνάρτηση MPI_CARTDIM_GET δίνει τον αριθμό των διαστάσεων ενός communicator με δομή πλέγματος**

  int MPI_Cartdim_get(MPI_Comm cart_comm, int* ndims)

- **Η ndims κρατάει τον αριθμό των διαστάσεων της καρτεσιανής εικονικής τοπολογίας.**

MPI_CARTDIM_GET

Variable Name	C Type	In/Out	Description
comm	MPI_Comm	Input	**communicator handle της καρτεσιανής τοπολογίας**
ndims	int *	Output	**αριθμός των διαστάσεων της τοπολογίας**

```
belongs[0] = 1;
belongs[1] = 0;
MPI_Cart_sub(grid_comm, belongs, &col_comm);

/* queries number of dimensions of cartesan grid */
MPI_Cartdim_get(col_comm, &ndims);
```

MPI_CART_GET

- **Η συνάρτηση MPI_CART_GET χρησιμοποιείται για να προσδιορισθούν ιδιότητες του communicator πλέγματος όπως η περιοδικότητα, ο αριθμός των διεργασιών/διάσταση, οι συντεταγμένες της καλούσας διεργασίας**

MPI_CART_GET

Μεταβλητή	Τύπος	In/Out	Περιγραφή
subgrid_comm	MPI_Comm	Input	Communicator handle
ndims	int	Input	**αριθμός των διαστάσεων της τοπολογίας**
dims	int *	Output	**πίνακας που περιέχει τον αριθμό των διεργασιών /διάσταση**
periods	int *	Output	**πίνακας που περιέχει ένα στοιχείο για κάθε διάσταση που δηλώνει αν είναι περιοδική ή όχι**
coords	int *	Output	**πίνακας που περιέχει τις καρτεσιανές συντεταγμένες της καλούσας διεργασίας**

MPI_CART_GET

```
dims[0] = nrow;
dims[1] = mcol;
MPI_Cart_create(MPI_COMM_WORLD, ndim, dims, period,
    reorder, &grid_comm);
MPI_Comm_rank(grid_comm, &me);
MPI_Cart_coords(grid_comm, me, ndim, coords);

/* create row subgrids */
belongs[0] = 1;
belongs[1] = 0;
MPI_Cart_sub(grid_comm, belongs, &row_comm);

/* Retrieve subgrid dimensions and other info */
MPI_Cartdim_get(row_comm, &mdims);
MPI_Cart_get(row_comm, mdims, dims, period, row_coords);
```

MPI_CART_GET

- **Often, MPI_CARTDIM_GET needs to be called first since ndims, the dimensions of the subgrid, is needed as input to MPI_CART_GET.**

0,0 (0)	*0,1 (1)*
1,0 (2)	*1,1 (3)*
2,0 (4)	*2,1 (5)*

Cartesian Grid

MPI_CART_SHIFT

- **Η MPI_CART_SHIFT χρησιμοποιείται για να προσδιορίσουμε δύο γείτονες της καλούσας διεργασίας κατά μήκος μιας διάστασης της καρτεσιανής τοπολογίας.**

```
int MPI_Cart_shift( MPI_Comm comm, int direction, int
        displ, int *source, int *dest )
```

- **Η διάσταση δίνεται στην παράμετρο εισόδου «direction». Οι δύο γείτονες επιστρέφονται στις παραμέτρους "source" και "destination" , και η απόσταση των δύο γειτόνων από την καλούσα διεργασία δίνεται στην παράμετρο εισόδου «displ».**

MPI_CART_SHIFT

Variable Name	C Type	In/Out	Description
comm	MPI_Comm	Input	Communicator handle
direction	int	Input	Η διάσταση κατά μήκος της οποίας θα λάβει χώρα το shift
displ	int	Input	shift (<0; >0; ή 0), ήτοι απόσταση από την καλούσα διεργασία
source	int *	Output	Η τάξη της διεργασίας source
dest	int *	Output	Η τάξη της διεργασίας destination

MPI_CART_SHIFT

```
dims[0] = nrow;     /* number of rows     */
dims[1] = mcol;     /* number of columns  */
periods[0] = 1;     /* cyclic in this direction */
periods[1] = 0;     /* no cyclic in this direction */
MPI_Cart_create(MPI_COMM_WORLD, ndim, dims, period,
    reorder, &comm2D);
MPI_Comm_rank(comm2D, &me);
MPI_Cart_coords(comm2D, me, ndim, coords);

index =  0;    /* shift along the 1st index (out of 2) */
displ =  1;    /* shift by  1 */
MPI_Cart_shift(comm2D, index, displ, &source, &dest);
```

MPI_CART_SHIFT

Αφού periods[0]=1, τότε οποιαδήποτε αναφορά πριν το πρώτο ή μετά το τελευταίο στοιχείο μιας στήλης θα ερμηνεύεται ως εξής:

- ***i = -1* αντιστοιχεί στο *i = 2*.**
- ***i = -2* αντιστοιχεί στο *i = 1*.**
- ***i = 3* αντιστοιχεί στο *i = 0*.**

	-1,0 (4)	*-1,1 (5)*	
0,-1(-1)	*0,0 (0)*	*0,1 (1)*	*0,2(-1)*
1,-1(-1)	*1,0 (2)*	*1,1 (3)*	*1,2(-1)*
2,-1(-1)	*2,0 (4)*	*2,1 (5)*	*2,2(-1)*
	3,0 (0)	*3,1 (1)*	

periods[0]=1;periods[1]=0

Αν ορίσουμε περιοδικότητα στις στήλες (periods[1]=1), τότε οποιαδήποτε αναφορά πριν το πρώτο ή μετά το τελευταίο στοιχείο μιας στήλης θα ερμηνεύεται ως εξής:

- ***j= -1* αντιστοιχεί στο *j= 1*.**
- ***j= 2* αντιστοιχεί στο *j= 0*.**

	-1,0 (-1)	*-1,1 (-1)*	
0,-1(1)	*0,0 (0)*	*0,1 (1)*	*0,2(0)*
1,-1(3)	*1,0 (2)*	*1,1 (3)*	*1,2(2)*
2,-1(5)	*2,0 (4)*	*2,1 (5)*	*2,2(4)*
	3,0 (-1)	*3,1 (-1)*	

periods[0]=0;periods[1]=1

MPI_CART_SHIFT

- **Αν η διεργασία που καλεί την MPI_CART_SHIFT είναι η 2, αφού index=0 και disp=1, τότε η τάξη της διεργασίας source και της διεργασίας destination (ως αποτέλεσμα του shift) θα είναι 0 και 4, αντίστοιχα.**

- **Αν η διεργασία που καλεί την MPI_CART_SHIFT είναι η 1, τότε η τάξη της διεργασίας source και της διεργασίας destination (ως αποτέλεσμα του shift) θα είναι 5 και 3, αντίστοιχα (η τάξη της διεργασίας source οφείλεται στο γεγονός ότι periods(0) = 1).**

MPI_CART_SHIFT

- Note:
  - Direction, the Cartesian grid dimension index, has range (0, 1, ..., ndim-1). For a 2D grid, the two choices for direction are 0 and 1.
  - MPI_CART_SHIFT is a query function. No action results from its call.
  - A negative returned value (MPI_UNDEFINED) of source or destination signifies the respective value is out of bound. It also implies that there is no periodicity along that direction.
  - If periodic condition is enabled along the shift direction, an out of bound does not result.
- Definition of MPI_UNDEFINED
  - A flag value returned by MPI when an integer value to be returned by MPI is not meaningfully defined.

MPI_CART_SHIFT Example

1. **Index=0, disp=+1, the source is the rank above the calling rank and the destination is the rank below it.**
2. **Index=0, disp= -2, the source is two ranks below the calling rank and the destination is two ranks above it.**
3. **Index=0, disp=+3, the source is three ranks above the calling rank and the destination is three rank below it.**
4. **Index=1, disp=+1, the source is the rank to the left of the calling rank and the destination is the rank to the right.**
5. **Index=1, disp=-1, the source is the rank to the right of the calling rank and the destination is the rank to the left.**

MPI_CART_SHIFT Example

```
#include "stdio.h"
#include "mpi.h"
void main(int argc, char *argv[])
{
    int nrow, mcol, irow, jcol, Iam, me, ndim;
    int p, ierr, root, index, displ;
    int source1, source2, source3, source4, source5;
    int dest1, dest2, dest3, dest4, dest5;

    int coords[2], dims[2];
    int periods[2], reorder;
    MPI_Comm comm2D;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &Iam);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
```

MPI_CART_SHIFT Example

```
nrow = 3; mcol = 2; ndim = 2;
root = 0; periods[0] = 1; periods[1] = 0; reorder = 1;

if (Iam == root)
{
        printf("       (       along the rows         )(   across columns    )\n");
        printf("         <== +1 ==>  <== -2 ==>  <== +3 ==>  <== +1 ==>  <== -1 ==>\n");
        printf("   2D   Row   Col  From    To  From    To  From    To  From    To  From
To\n");
        printf(" Rank    i    j   Src  Dest   Src  Dest   Src  Dest   Src  Dest   Src  Dest\n");
}
MPI_Barrier(MPI_COMM_WORLD);

/* create cartesian topology for processes */
dims[0] = nrow;                         /* number of rows */
dims[1] = mcol;                         /* number of columns */
MPI_Cart_create(MPI_COMM_WORLD, ndim, dims, periods, reorder, &comm2D);
MPI_Comm_rank(comm2D, &me);
MPI_Cart_coords(comm2D, me, ndim, coords);
```

MPI_CART_SHIFT Example

```
index = 0;       /* shift along the 1st direction (0; not 1) */
displ = 1;       /* shift by  1 */
MPI_Cart_shift(comm2D, index, displ, &source1, &dest1);
index = 0;       /* shift along the 1st direction */
displ = -2;      /* shift by -2 */
MPI_Cart_shift(comm2D, index, displ, &source2, &dest2);
index = 0;       /* shift along the 1st direction (0; not 1) */
displ = 3;       /* shift by  3 */
MPI_Cart_shift(comm2D, index, displ, &source3, &dest3);
index = 1;       /* shift along the 2nd direction (1; not 2) */
displ = 1;       /* shift by  1 */
MPI_Cart_shift(comm2D, index, displ, &source4, &dest4);
index = 1;       /* shift along the 2nd direction */
displ = -1;      /* shift by -1 */
MPI_Cart_shift(comm2D, index, displ, &source5, &dest5);

printf("%5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n", me,
coords[0], coords[1], source1, dest1, source2, dest2, source3, dest3, source4,
dest4, source5, dest5);
MPI_Finalize();            }
```

MPI_CART_SHIFT Example

- **Output**
  - **Note that some of the returned ranks in the last four columns are negative because they are out of bounds and are assigned the value MPI_UNDEFINED. The value of MPI_UNDEFINED is implementation dependent. In this case, it is -1.**

```
$ mpirun -np 6 _cart_shift_example
                    (          along the rows           )(   across columns     )
                     <== +1 ==>  <== -2 ==>  <== +3 ==>  <== +1 ==>  <== -1 ==>
   2D    Row    Col   From    To   From    To   From    To   From    To   From    To
 Rank     i      j    Src  Dest    Src  Dest    Src  Dest    Src  Dest    Src  Dest
    0     0      0      4     2      4     2      0     0     -1     1      1    -1
    1     0      1      5     3      5     3      1     1      0    -1     -1     0
    2     1      0      0     4      0     4      2     2     -1     3      3    -1
    3     1      1      1     5      1     5      3     3      2    -1     -1     2
    4     2      0      2     0      2     0      4     4     -1     5      5    -1
    5     2      1      3     1      3     1      5     5      4    -1     -1     4
```

MPI_CART_SHIFT Example

	-3,0(0)	-3,0(1)	
	-2,0(2)	-2,1(3)	
	-1,0(4)	-1,1(5)	
0,-1(-1)	0,0(0)	0,1(1)	0,2(-1)
1,-1(-1)	1,0(2)	1,1,(3)	1,2(-1)
2,-1(-1)	2,0(4)	2,1(5)	2,2(-1)
	3,0(0)	3,1(1)	
	4,0(2)	4,1,(3)	
	5,0(4)	5,1(5)	

periods[0]=1;periods[1]=0

Παράδειγμα (διαχείριση και επικοινωνία σε τοπολογία 4x4)

- **Να οργανώσουμε το σύνολο των διεργασιών σε ιδεατή καρτεσιανή τοπολογία δύο διαστάσεων 4x4, και στη συνέχεια να γράψουμε τον απαραίτητο κώδικα για να υπολογίζεται από κάθε διεργασία το άθροισμα των στοιχείων των τεσσάρων γειτονικών της διεργασιών.**

- **Για να επιτευχθεί αυτό θα πρέπει κάθε διεργασία να στέλνει και να παραλαμβάνει ένα μήνυμα προς και από κάθε μία από τις τέσσερις γειτονικές της.**

```
#include <stdio.h>
#include "mpi.h"
#define SIZE 16
#define UP    0
#define DOWN  1
#define LEFT  2
#define RIGHT 3

int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, source, dest, outbuf, i, tag=1;

int inbuf[4]={MPI_PROC_NULL,MPI_PROC_NULL,MPI_PROC_NULL,
            MPI_PROC_NULL};
int nbrs[4], dims[2]={4,4};
int periods[2]={1,1}, coords[2];
int reorder=0, sum=0;
```

```
/* ορίζουμε αντικείμενα Request και Status για τον τελικό έλεγχο της ολοκλήρωσης του συνόλου των επικοινωνιών */

MPI_Request reqs[8];
MPI_Status stats[8];
MPI_Comm cartcomm;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder,
&cartcomm);

MPI_Comm_rank(cartcomm, &rank);
MPI_Cart_coords(cartcomm, rank, 2, coords);
MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP], &nbrs[DOWN]);
MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT], &nbrs[RIGHT]);

/* Αρχικοποιούμε το στοιχείο κάθε διεργασίας να έχει τιμή ίση με την τάξη της */
outbuf = rank;
```

```
for (i=0; i<4; i++) {
  dest = nbrs[i];
  source = nbrs[i];
  MPI_Isend(&outbuf,1,MPI_INT,dest,tag,MPI_COMM_WORLD,
            &reqs[i]);
  MPI_Irecv(&inbuf[i],1,MPI_INT,source,tag,
            MPI_COMM_WORLD, &reqs[i+4]);
}

/* Ελέγχουμε αν ολοκληρώθηκαν επιτυχώς όλες οι επικοινωνίες */
 MPI_Waitall(8, reqs, stats);

for (i=0; i<4; i++)
  sum += inbuf[i];

printf("rank= %d coords= %d %d  neighbors(u,d,l,r)= %d %d %d %d
sum = %d\n", rank, coords[0], coords[1], nbrs[UP], nbrs[DOWN],
nbrs[LEFT], nbrs[RIGHT], sum);

MPI_Finalize();   }
```