Lectia 7. RUTINE MPI DE ADMINISTRARE A COMUNICATOARELOR ȘI GRUPELOR DE PROCESE.

7.1 Grupe și comunicatori

O grupă este o mulțime ordonată de procese. Fiecare proces dintr-un grup este asociat unui rang întreg unic. Valorile rangului pornesc de la 0 și merg până la $N - 1$, unde N este numărul de procese din grup.

În MPI un grup este reprezentat în memoria sistemului ca un obiect. El este accesibil programatorului numai printr-un „handle". Există două grupe prestabilite: **`MPI_GROUP_EMPTY`** – grupul care nu conține niciun proces și **`MPI_GROUP_NUL`** – se returnează această valoare în cazul când grupul nu poate fi creat. Un grup este totdeauna asociat cu un obiect comunicator. Un comunicator cuprinde un grup de procese care pot comunica între ele. Toate mesajele MPI trebuie să specifice un comunicator. În sensul cel mai simplu, comunicatorul este o „etichetă" suplimentară care trebuie inclusă în apelurile MPI. Ca și grupele, comunicatorii sunt reprezentați în memoria sistemului ca obiecte și sunt accesibili programatorului numai prin „handles". De exemplu, un handle pentru comunicatorul care cuprinde toate task-urile este **`MPI_COMM_WORLD`**. Rutinele grupului sunt utilizate în principal pentru a specifica procesele care trebuie utilizate pentru a construi un comunicator.

Scopurile principale ale obiectelor grup și comunicator:

- Permit organizarea task-urilor, pe baza unor funcții, în grupuri de task-uri.
- Abilitează operațiile de comunicare colectivă într-un subset de task-uri într-un fel în relație.
- Asigură baza pentru implementarea topologiilor virtuale definite de utilizator.
- Garantează siguranța comunicării.

Restricții și alte considerații asupra programării:

- Grupurile/comunicatorii sunt obiecte dinamice – pot fi create și distruse în timpul execuției programului.
- Procesele pot fi în mai mult de un grup/comunicator. Ele vor avea un rang unic în fiecare grup/comunicator.

În MPI comunicarea poate avea loc în cadrul unui grup (intracomunicare) sau între două grupuri distincte (intercomunicare). Corespunzător, comunicatorii se împart în două categorii: **intracomunicatori** și **intercomunicatori**. Un intracomunicator descrie:

- ✓ un grup de procese;
- ✓ contexte pentru comunicare punct la punct și colectivă (aceste două contexte sunt disjuncte, astfel că un mesaj punct la punct nu se poate confunda cu un mesaj colectiv, chiar dacă au același tag);
- ✓ o topologie virtuală (eventual);
- ✓ alte atribute.

MPI are o serie de operații pentru manipularea grupurilor, printre care:.

- ✓ aflarea grupului asociat unui comunicator;
- ✓ găsirea numărului de procese dintr-un grup și a rangului procesului apelant;
- ✓ compararea a două grupuri;
- ✓ reuniunea, intersecția, diferența a două grupuri;
- ✓ crearea unui nou grup din altul existent, prin includerea sau excluderea unor procese;
- ✓ desființarea unui grup.

Funcții similare sunt prevăzute și pentru manipularea intracomunicatorilor:

- ✓ găsirea numărului de procese sau a rangului procesului apelant;
- ✓ compararea a doi comunicatori;
- ✓ duplicarea, crearea unui comunicator;
- ✓ partiționarea grupului unui comunicator în grupuri disjuncte;
- ✓ desființarea unui comunicator.

Un intercomunicator leagă două grupuri, împreună cu contextele de comunicare partajate de cele două grupuri. Contextele sunt folosite doar pentru operații punct la punct, neexistând comunicare colectivă intergrupuri. Nu există topologii virtuale în cazul intercomunicării. O intercomunicare are loc între un

proces inițiator, care aparține unui grup local, și un proces țintă, care aparține unui grup distant. Procesul țintă este specificat printr-o pereche (comunicator, rang) relativă la grupul distant. MPI conține peste 40 de rutine relative la grupuri, comunicatori și topologii virtuale. Mai jos vom descrie o parte din aceste funcții (rutine).

7.2 Funcțiile MPI de gestionare a grupelor de procese

Funcția MPI_Group_size

Această funcție permite determinarea numărului de procese din grup. Prototipul acestei funcții în limbajul C++ este

```
int MPI_Group_size(MPI_Group group, int *size)
```

unde

IN **group** – numele grupei;
OUT **size** – numărul de procese din grup.

Funcția MPI_Group_rank

Această funcție permite determinarea rankului (un identificator numeric) al proceselor din grup. Prototipul acestei funcții în limbajul C++ este

```
int MPI_Group_rank(MPI_Group group, int *rank)
```

unde

IN **group** – numele grupei;
OUT **rank** – numărul procesului din grup.

Dacă procesul nu face parte din grupul indicat, atunci se returnează valoarea **MPI_UNDEFINED.**

Funcția MPI_Comm_group

Această funcție permite crearea unui grup de procese prin intermediul unui comunicator. Prototipul acestei funcții în limbajul C++ este

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)
```

unde

IN **comm** – numele comunicatorului;
OUT **group** – numele grupului.

Astfel, se creează grupul cu numele **group** pentru mulțimea de procese care aparțin comunicatorului cu numele **comm.**

Funcțile MPI_Comm_union, MPI_Comm_intersection, MPI_Comm_difference

Următoarele trei funcții au aceeași sintaxă și se utilizează pentru crearea unui grup nou de procese MPI , ca rezultat al unor operații asupra mulțimilor de procese din două grupe. Prototipul acestor funcții în limbajul C++ este

```
int MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group
    *newgroup)
int MPI_Group_intersection(MPI_Group group1, MPI_Group group2,
    MPI_Group *newgroup)
int MPI_Group_difference(MPI_Group group1, MPI_Group group2,
    MPI_Group *newgroup)
```

unde

IN **group1** – numele primului grup de procese;
IN **group2** – numele grupului doi de procese;
OUT **newgroup** – numele grupului nou creat.

Operațiunile sunt definite după cum urmează:

Union – creează un nou grup din procesele grupei **group1** și din acele procese ale grupei **group2** care nu aparțin grupei **group1** (operația de *reuniune*).

Intersection – creează un nou grup din procesele grupei **group1** care aparțin și grupei **group2** (operația de *intersecție*).

Difference – creează un nou grup din procesele grupei **group1** care nu aparțin grupei **group2** (operația de *diferență*).

Noul grup poate fi și vid, atunci se returnează **MPI_GROUP_EMPTY**.

Funcțile MPI_Comm_incl, MPI_Comm_excl

Următoarele două funcții sunt de aceeași sintaxă, dar sunt complementare. Prototipul acestor funcții în limbajul C++ este

```
int MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group
    *newgroup)
int MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group
    *newgroup)
```

unde

IN **group**	– numele grupei existente („părinte”);
IN **n**	– numărul de elemente în vectorul **ranks** (care este de fapt egal cu numărul de procese din grupul nou creat);
IN **ranks**	– un vector ce conține rankurile proceselor;
OUT **newgroup**	– numele grupului nou creat.

Funcția **MPI_Group_incl** creează un nou grup, care constă din procesele din grupul **group**, enumerate în vectorul **ranks**. Procesul cu numărul *i* în noul grup **newgroup** este procesul cu numărul **ranks[i]** din grupul existent **group**.

Funcția **MPI_Group_excl** creează un nou grup din acele procese ale grupului existent **group** care nu sunt enumerate în vectorul **ranks**. Procesele din grupul **newgroup** sunt ordonate la fel ca și în grupul inițial **group**.

Vom exemplifica rutinele MPI descrise mai sus.

Exemplul 7.1 *Fie dat un grup „părinte” de procese MPI numerotate 0,...,size-1. Să se elaboreze un program MPI în care se creează un nou grup de k=size/2 procese, alegând aleator procese din grupul „părinte”. Fiecare proces din grupul creat tipărește informația despre sine în forma: rankul din grupul nou (rankul din grupul părinte).*

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele menționate în exemplul 7.1

```
#include<stdio.h>
#include <stdio.h>
#include <mpi.h>
int main(int argc,char *argv[])
{
int i,k,p,size,rank;
int rank_gr;
 char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Status status;
MPI_Group MPI_GROUP_WORLD, newgr;
int* ranks;
int namelen;
MPI_Init(&argc,&argv);
```

```
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Get_processor_name(processor_name,&namelen);
if (rank ==0)
printf("\n=====REZULTATUL PROGRAMULUI '%s' \n",argv[0]);
MPI_Barrier(MPI_COMM_WORLD);
srand(time(0));
k=size / 2;
ranks = (int*)malloc(k*sizeof(int));
int rN = 0;
int repeat;
      for (i = 0; i < k; i++)
      {
          do
          {
          repeat = 0;
          rN = rand() % size;
          for (int j = 0; j < i; ++j)
          {
          if (rN == ranks[j])
                    {
                    repeat = 1;
                    break;
                    }
          }
          } while(repeat == 1);

         ranks[i] = rN;
      }
      if(rank==0)
      {
      printf("Au fost extrase aleator %d numere dupa cum urmeaza:\n",k);
          for (i = 0; i < k; i++)
          printf(" %d ",ranks[i]);
          printf(" \n");
      MPI_Barrier(MPI_COMM_WORLD);
      }
      else MPI_Barrier(MPI_COMM_WORLD);
      MPI_Comm_group(MPI_COMM_WORLD,&MPI_GROUP_WORLD);
      MPI_Group_incl(MPI_GROUP_WORLD,k,ranks,&newgr);
      MPI_Group_rank(newgr,&rank_gr);
      if (rank_gr != MPI_UNDEFINED)
      printf ("Sunt procesul cu rankul %d (%d) de pe nodul %s. \n", rank_gr, rank, processor_name);
MPI_Group_free(&newgr);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_5_1.exe Exemplu _3_5_1.cpp[1]
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 16 -machinefile ~/nodes
HB_Grup_Proc_Aleatoare.exe
=====REZULTATUL PROGRAMULUI 'Exemplu _3_5_1.exe'
Au fost extrase aleator 8 numere dupa cum urmeaza:
 8  2  6  1  14  3  10  13
Sunt procesul cu rankul 2 (6) de pe nodul compute-0-4.local.
Sunt procesul cu rankul 7 (13) de pe nodul compute-0-8.local.
Sunt procesul cu rankul 0 (8) de pe nodul compute-0-6.local.
Sunt procesul cu rankul 3 (1) de pe nodul compute-0-2.local.
Sunt procesul cu rankul 4 (14) de pe nodul compute-0-8.local.
Sunt procesul cu rankul 6 (10) de pe nodul compute-0-6.local.
Sunt procesul cu rankul 1 (2) de pe nodul compute-0-2.local.
Sunt procesul cu rankul 5 (3) de pe nodul compute-0-2.local.
[Hancu_B_S@hpc]$
```

7.3 Funcțiile MPI de gestionare a comunicatoarelor

În acest paragraf vom analiza funcțiile de lucru cu comunicatorii. Acestea sunt împărțite în funcții de acces la comunicator și funcții utilizate pentru a crea un comunicator. Funcțiile de acces sunt locale și nu necesită comunicații, spre deosebire de funcțiile de creare a comunicatoarelor, care sunt colective și pot necesita comunicații interprocesor. Două funcții de acces la comunicator **`MPI_Comm_size`** și **`MPI_Comm_rank`** au fost analizate deja mai sus.

Funcția MPI_Comm_compare

Este utilizată pentru a compara două comunicatoare. Prototipul funcției în limbajul C++ este

```
int MPI_Comm_compare(MPI_Comm comm1,MPI_Comm comm2, int *result)
```

unde

IN **comm1**	– numele primului comunicator;
IN **comm2**	– numele comunicatorului al doilea;
OUT **result**	– rezultatul comparației.

Valorile posibile ale variabilei **`result`**:

`MPI_IDENT`	Comunicatoarele sunt identice, reprezintă același mediu de comunicare.
`MPI_CONGRUENT`	Comunicatoarele sunt congruente, două medii de comunicare cu aceeași parametri de grup.
`MPI_SIMILAR`	Comunicatoarele sunt similare, grupele conțin aceleași procese, dar cu o altă distribuire a rankurilor.
`MPI_UNEQUAL`	Toate celelalte cazuri.

Crearea unor noi medii de comunicare, comunicatoare, se poate face cu una din următoarele funcții: **`MPI_Comm_dup, MPI_Comm_create, MPI_Comm_split.`**

Funcția MPI_Comm_dup

Crearea unui comunicator ca și copie a altuia. Prototipul funcției în limbajul C++ este

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm))
```

[1] Numele programului corespunde numelui exemplului din notele de curs Boris HÎNCU, Elena CALMÎȘ "MODELE DE PROGRAMARE PARALELĂ PE CLUSTERE. PARTEA I. PROGRAMARE MPI". Chisinau 2016.

unde

IN **comm** – numele comunicatorului părinte;
OUT **newcomm** – numele comunicatorului nou creat.

Funcția MPI_Comm_create

Funcția este utilizată pentru crearea unui comunicator pentru un grup de procese. Prototipul funcției în limbajul C++ este

```
int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm
    *newcomm)
```

unde

IN **comm** – numele comunicatorului părinte;
IN **group** – numele grupei;
OUT **newcomm** – numele comunicatorului nou creat.

Pentru procesele care nu fac parte din grupul **group** se returnează valoarea **MPI_COMM_NULL**. Funcția va returna un cod de eroare dacă **group** nu este un subgrup al comunicatorului părinte.

Vom exemplifica rutinele MPI pentru gestionarea comunicatoarelor descrise mai sus.

Exemplul 7.2 *Să se elaboreze un program MPI în limbajul C++ în care se creează un nou grup de procese care conține câte un singur proces de pe fiecare nod al clusterului. Pentru acest grup de procese să se creeze un comunicator. Fiecare proces din comunicatorul creat tipărește informația despre sine în forma: rankul din comunicatorul nou (rankul din comunicatorul părinte).*

Elaborarea comunicatoarelor de acest tip poate fi utilizat pentru a exclude transmiterea mesajelor prin rutine MPI între procese care aparțin aceluiași nod. Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele menționate în exemplul 7.2.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
int i, p, k=0, size, rank, rank_new;
int Node_rank;
int Nodes; //numarul  de noduri
int local_rank = atoi(getenv("OMPI_COMM_WORLD_LOCAL_RANK"));
char processor_name[MPI_MAX_
    PROCESSOR_NAME];
MPI_Status status;
MPI_Comm com_new, ring1;
MPI_Group MPI_GROUP_WORLD, newgr;
int *ranks,*newGroup;
int namelen;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(processor_name, &namelen);
if (rank == 0) {
printf("=====REZULTATUL PROGRAMULUI '%s' \n", argv[0]);
printf ("Rankurile proceselor din comuni
    catorului 'MPI_COMM_WOLD' au fost repartizate astfel: \n"); }
MPI_Barrier(MPI_COMM_WORLD);
 // Se determina numarul de noduri (egal cu  numarul de procese în grupul creat
```

```
printf ("Rankul %d de pe nodul %s. \n",rank, processor_name);
if (local_rank == 0) k = 1;
MPI_Allreduce(&k, &Nodes, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
newGroup=(int *)malloc(Nodes*sizeof(int));
ranks = (int *) malloc(size * sizeof(int));
int r;
// Se construieste vectorul newGroup
  if (local_rank == 0)
      ranks[rank] = rank;
  else
      ranks[rank] = -1;
  for (int i = 0; i < size; ++i)
      MPI_Bcast(&ranks[i], 1, MPI_INT, i, MPI_COMM_WORLD);
  for (int i = 0, j = 0; i < size; ++i)
  {
      if (ranks[i] != -1)
      {
          newGroup[j] = ranks[i];
          ++j;
      }
  }
MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
MPI_Group_incl(MPI_GROUP_WORLD, Nodes, newGroup, &newgr);
MPI_Comm_create(MPI_COMM_WORLD, newgr, &com_new);
MPI_Group_rank(newgr, &rank_new);
if (rank_new!= MPI_UNDEFINED)
printf ("Procesul cu rankul %d al com. 'com_new' (%d com. 'MPI_COMM_WOLD') de pe nodul %s. \n",
      rank_new,rank,processor_name);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu_3_5_2.exe Exemplu_3_5_2.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 28 -host compute-0-1,compute-0-2,compute-0-
4,compute-0-6,compute-0-8,compute-0-9,compute-0-10 Exemplu_3_5_2.exe
=====REZULTATUL PROGRAMULUI 'Exemplu_3_5_2.exe'
Rankurile proceselor din comunicatorului 'MPI_COMM_WOLD' au fost repartizate astfel:
Rankul 18 de pe nodul compute-0-8.local.
Rankul 19 de pe nodul compute-0-9.local.
Rankul 14 de pe nodul compute-0-1.local.
Rankul 6 de pe nodul compute-0-10.local.
Rankul 22 de pe nodul compute-0-2.local.
Rankul 2 de pe nodul compute-0-4.local.
Rankul 10 de pe nodul compute-0-6.local.
Rankul 11 de pe nodul compute-0-8.local.
Rankul 12 de pe nodul compute-0-9.local.
Rankul 0 de pe nodul compute-0-1.local.
Rankul 20 de pe nodul compute-0-10.local.
```

```
Rankul 8 de pe nodul compute-0-2.local.
Rankul 9 de pe nodul compute-0-4.local.
Rankul 3 de pe nodul compute-0-6.local.
Rankul 25 de pe nodul compute-0-8.local.
Rankul 26 de pe nodul compute-0-9.local.
Rankul 7 de pe nodul compute-0-1.local.
Rankul 27 de pe nodul compute-0-10.local.
Rankul 15 de pe nodul compute-0-2.local.
Rankul 16 de pe nodul compute-0-4.local.
Rankul 17 de pe nodul compute-0-6.local.
Rankul 4 de pe nodul compute-0-8.local.
Rankul 5 de pe nodul compute-0-9.local.
Rankul 21 de pe nodul compute-0-1.local.
Rankul 13 de pe nodul compute-0-10.local.
Rankul 1 de pe nodul compute-0-2.local.
Rankul 23 de pe nodul compute-0-4.local.
Rankul 24 de pe nodul compute-0-6.local.
Pprocesul cu rankul 3 al com. 'com_new'(3 com. 'MPI_COMM_WOLD') de pe nodul compute-0-6.local.
Procesul cu rankul 5 al com. 'com_new'(5 com. 'MPI_COMM_WOLD') de pe nodul compute-0-9.local.
Procesul cu rankul 6 al com. 'com_new'(6 com. 'MPI_COMM_WOLD') de pe nodul compute-0-10.local.
Procesul cu rankul 1 al com. 'com_new'(1 com. 'MPI_COMM_WOLD') de pe nodul compute-0-2.local.
Procesul cu rankul 2 al com. 'com_new'(2 com. 'MPI_COMM_WOLD') de pe nodul compute-0-4.local.
Procesul cu rankul 4 al com. 'com_new'(4 com. 'MPI_COMM_WOLD') de pe nodul compute-0-8.local.
Procesul cu rankul 0 al com. 'com_new'(0 com. 'MPI_COMM_WOLD') de pe nodul compute-0-1.local.
[Hancu_B_S@hpc]$
```

7.4 Topologii virtuale. Rutine de creare a topologiei carteziene

În termenii MPI, o topologie virtuală descrie o aplicație/ordonare a proceselor MPI într-o „formă” geometrică. Cele două tipuri principale de topologii suportate de MPI sunt cea *carteziană (grilă)* și cea sub formă *de graf*. Topologiile MPI sunt virtuale – poate să nu existe nicio relație între structura fizică a unei mașini paralele și topologia proceselor. Topologiile virtuale sunt construite pe grupuri și comunicatori MPI și trebuie să fie programate de cel care dezvoltă aplicația. Topologiile virtuale pot fi utile pentru aplicații cu forme de comunicare specifice – forme (patterns) care se potrivesc unei structuri topologice MPI. De exemplu, o topologie carteziană se poate dovedi convenabilă pentru o aplicație care reclamă comunicare cu *4* din vecinii cei mai apropiați pentru date bazate pe grile.

Funcția MPI_Cart_create

Pentru a crea un comunicator (un mediu virtual de comunicare) cu topologie carteziană este folosită rutina **`MPI_Cart_create`**. Cu această funcție se poate crea o topologie cu număr arbitrar de dimensiuni, și pentru fiecare dimensiune în mod izolat pot fi aplicate condiții – limită periodice. Astfel, în funcție de care condiții la limită se impun, pentru o topologie unidimensională obținem sau structură liniară, sau un inel (cerc). Pentru topologie bidimensională, respectiv, fie un dreptunghi sau un cilindru sau tor. Prototipul funcției în limbajul C++ este

```
int MPI_Cart_create(MPI_Comm comm_old,int ndims,int *dims,int
    *periods,int reorder,MPI_Comm *comm_cart)
```

unde

IN **`comm_old`** – numele comunicatorului părinte;
IN **`ndims`** – dimensiunea (numărul de axe);
IN **`dims`** – un vector de dimensiunea **`ndims`** în care se indică numărul de procese pe

fiecare axă;

IN **periods** – un vector logic de dimensiunea **ndims** în care se indică condițiile la limită pentru fiecare axă (**true-** condițiile sunt periodice, adică axa este „închisă”, **false-** condițiile sunt neperiodice, adică axa nu este „închisă”);

IN **reorder** – o variabilă logică, care indică dacă se va face renumerotarea proceselor (**true**) sau nu (**false**);

OUT **comm_cart** – numele comunicatorului nou creat.

Funcția este colectivă, adică trebuie să fie executată pe toate procesele din grupul de procese ale comunicatorului **comm_old.** Parametrul **reorder=false** indică faptul că ID-urile proceselor din noul grup vor fi aceleași ca și în grupul vechi. Dacă **reorder=true**, MPI va încerca să le schimbe cu scopul de a optimiza comunicarea.

Următoarele funcții descrise în acest paragraf au un caracter auxiliar sau informațional.

Funcția MPI_Dims_create

Această funcție se utilizează pentru determinarea unei configurații optimale ale rețelei de procese. Prototipul funcției în limbajul C++ este

```
int MPI_Dims_create(int nnodes, int ndims, int *dims)
```

unde

IN **nnodes** – numărul total de noduri în rețea;

IN **ndims** – dimensiunea (numărul de axe);

IN/OUT **dims** – un vector de dimensiunea **ndims** în care se indică numărul recomandat de procese pe fiecare ax.

La intrare în procedură în vectorul **dims** trebuie să fie înregistrate numere întregi non-negative. În cazul în care elementul **dims[i]** este un număr pozitiv, atunci pentru această axă (direcție) nu este realizat niciun calcul (numărul de procese de-a lungul acestei direcții este considerat a fi specificat). Se determină (calculează) numai acele **dims[i]**, care înainte de aplicarea procedurii sunt setate la *0*. Funcția are ca scop crearea unei distribuții cât mai uniforme a proceselor de-a lungul axei, aranjându-le în ordine descrescătoare. De exemplu, pentru 12 procese, aceasta va construi o grilă tridimensională *4×3×1*. Rezultatul acestei rutine poate fi folosită ca un parametru de intrare pentru funcția **MPI_Cart_create**.

Funcția MPI_Cart_get

Această funcție se utilizează pentru a obține o informație detaliată despre comunicatorul cu topologie carteziană. Prototipul funcției în limbajul C++ este

```
int MPI_Cart_get(MPI_Comm comm, int ndims, int *dims, int *periods,
    int *coords)
```

unde

IN **comm** – comunicatorul cu topologie carteziană;

IN **ndims** – dimensiunea (numărul de axe);

OUT **dims** – un vector de dimensiunea **ndims** în care se returnează numărul de procese pe fiecare axă;

OUT **periods** – un vector logic de dimensiunea **ndims** în care se returnează condițiile la limită

pentru fiecare axă (**true-** condițiile sunt periodice, **false-** condițiile sunt neperiodice);

OUT **coords** – coordonatele carteziene ale proceselor care apelează această funcție.

Următoarele două funcții realizează corespondența dintre rankul (identificatorul) procesului în comunicatorul pentru care s-a creat topologia carteziană și coordonatele sale într-o grilă carteziană.

Funcția MPI_Cart_rank

Această funcție se utilizează pentru a obține rankul proceselor în baza coordonatelor sale. Prototipul funcției în limbajul C++ este

```
int MPI_Cart_rank(MPI_Comm comm, int *coords, int *rank)
```

unde

IN **comm** – comunicatorul cu topologie carteziană;

IN **coords** – coordonatele în sistemul cartezian);

OUT **rank** – rankul (identificatorul) procesului.

Funcția MPI_Cart_coords

Această funcție se utilizează pentru a obține coordonatele carteziene ale proceselor în baza rankurilor (identificatoarelor) sale. Prototipul funcției în limbajul C++ este

```
int MPI_Cart_coords(MPI_Comm comm, int rank, int ndims, int *coords)
```

unde

IN **comm** – comunicatorul cu topologie carteziană;

IN **rank** – rankul (identificatorul) procesului;

IN **ndims** – numărul de axe (direcții);

OUT **coords** – coordonatele în sistemul cartezian.

Funcția MPI_Cart_shift

În mulți algoritmi numerici se utilizează operația de deplasare a datelor de-a lungul unor axe ale grilei carteziene. În MPI există funcția **MPI_Cart_shift** care realizează această operație. Mai precis, deplasarea datelor este realizată folosind **MPI_Sendrecv**, iar funcția **MPI_Cart_shift** calculează pentru fiecare proces parametrii funcției **MPI_Sendrecv funcția** (sursa și destinația). Prototipul funcției în limbajul C++ este

```
int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int
    *rank_source, int *rank_dest)
```

unde

IN **comm** – comunicatorul cu topologie carteziană;

IN **direction** – numărul axei (direcției) de-a lungul căreia se realizează deplasarea;

IN **disp** – valoarea pasului de deplasare (poate fi pozitivă – deplasare în direcția acelor cronometrului, sau negative – deplasare în direcția inversă acelor cronometrului);

OUT – rankul procesului de la care se vor

rank_source recepționa datele;
OUT **rank_dest** – rankul procesului care va recepționa datele.

Vom exemplifica rutinele MPI pentru gestionarea comunicatoarelor cu topologie carteziană.

Exemplul 7.3 *Să se elaboreze un program MPI în limbajul C++ în care se creează un nou grup de procese care conține câte un singur proces de pe fiecare nod al clusterului. Pentru acest grup de procese să se creeze un comunicator cu topologie carteziană de tip cerc (inel) și să se realizeze transmiterea mesajelor pe cerc.*

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 7.3.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
int i, p, k=0, size, size_new, rank, rank_new, sour, dest;
int Node_rank, rank_gr;
int Nodes;
int local_rank = atoi(getenv("OMPI_COMM_WORLD_LOCAL_RANK"));
char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Status status;
MPI_Comm com_new, ring1;
MPI_Group MPI_GROUP_WORLD, newgr;
int dims[1], period[1], reord;
int *ranks,*newGroup;
int namelen;
int D1 = 123, D2;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(processor_name, &namelen);
 if (rank == 0)
  printf("=====REZULTATUL PROGRAMULUI '%s' \n", argv[0]);
   MPI_Barrier(MPI_COMM_WORLD);
   if (local_rank == 0) k = 1;
   MPI_Allreduce(&k, &Nodes, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
   newGroup=(int *) malloc(Nodes * sizeof(int));
   ranks = (int *) malloc(size * sizeof(int));
    int r;
   if (local_rank == 0)
      ranks[rank] = rank;
   else
      ranks[rank] = -1;

   for (int i = 0; i < size; ++i)
      MPI_Bcast(&ranks[i], 1, MPI_INT, i, MPI_COMM_WORLD);
    for (int i = 0, j = 0; i < size; ++i)
   {
```

```
      if (ranks[i] != -1)
      {
         newGroup[j] = ranks[i];
         ++j;
      }
   }
MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
MPI_Group_incl(MPI_GROUP_WORLD, Nodes, newGroup, &newgr);
MPI_Comm_create(MPI_COMM_WORLD, newgr, &com_new);
MPI_Group_rank(newgr, &rank_gr);
 if (rank_gr != MPI_UNDEFINED) {
      MPI_Comm_size(com_new, &size_new);
      MPI_Comm_rank(com_new, &rank_new);
      dims[0] = size_new;
      period[0] = 1;
      reord = 1;
      MPI_Cart_create(com_new, 1, dims, period, reord, &ring1);
      MPI_Cart_shift(ring1, 1, 2, &sour, &dest);
      D1 = D1 + rank;
      MPI_Sendrecv(&D1, 1, MPI_INT, dest, 12, &D2, 1, MPI_INT, sour, 12, ring1, &status);
      if (rank_new == 0) {
     printf("===Rezultatul MPI_Sendrecv:\n");
        MPI_Barrier(com_new);
      }
      else MPI_Barrier(com_new);
printf ("Proc. %d (%d from %s), recv. from proc. %d the value %d and send to proc. %d the value %d\n",
      rank_new, rank,processor_name, sour, D2,dest,D1);
      MPI_Barrier(com_new);
      MPI_Group_free(&newgr);
      MPI_Comm_free(&ring1);
      MPI_Comm_free(&com_new);
   }
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu_3_5_3.exe Exemplu_3_5_3.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 44 -machinefile ~/nodes o Exemplu_3_5_3.exe
=====REZULTATUL PROGRAMULUI o Exemplu_3_5_3.exe
===Rezultatul MPI_Sendrecv:
Proc.0 (0 from compute-0-0.local), recv. from proc. 9 the value 159 and send to proc. 2 the value 123
Proc.1 (4 from compute-0-1.local), recv. from proc. 10 the value 163 and send to proc. 3 the value 127
Proc.5 (20 from compute-0-6.local), recv. from proc. 3 the value 135 and send to proc. 7 the value 143
Proc.2 (8 from compute-0-2.local), recv. from proc. 0 the value 123 and send to proc. 4 the value 131
Proc.3 (12 from compute-0-4.local), recv. from proc. 1 the value 127 and send to proc. 5 the value 135
Proc.4 (16 from compute-0-5.local), recv. from proc. 2 the value 131 and send to proc. 6 the value 139
Proc.7 (28 from compute-0-8.local), recv. from proc. 5 the value 143 and send to proc. 9 the value 151
Proc.6 (24 from compute-0-7.local), recv. from proc. 4 the value 139 and send to proc. 8 the value 147
```

```
Proc.8 (32 from compute-0-9.local), recv. from proc. 6 the value 147 and send to proc. 10 the value 155
Proc.9 (36 from compute-0-10.local), recv. from proc. 7 the value 151 and send to proc. 0 the value 159
Proc.10 (40 from compute-0-12.local), recv. from proc. 8 the value 155 and send to proc. 1 the value 163
[Hancu_B_S@hpc]$
```

Exemplul 7.4 *Să se elaboreze un program MPI în limbajul C++ în care se creează un comunicator cu topologie carteziană de tip paralelepiped și să se determine pentru un anumit proces vecinii săi pe fiecare axă de coordonate.*

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 7.4.

```
#include <mpi.h>
#include <stdio.h>
#include <iostream>
#include <vector>
using namespace std;
int main(int argc, char *argv[])
{
int rank,test_rank=2;
int size;
int ndims = 3;
int source, dest;
int up_y,right_y,right_x,left_x,up_z, down_z;
int dims[3]={0,0,0},coords[3]={0,0,0};
int coords_left_x[3]={0,0,0}, coords_right_x[3]={0,0,0},    coords_left_y[3]={0,0,0},
    coords_right_y[3]={0,0,0},  coords_up_z[3]={0,0,0}, coords_down_z[3]={0,0,0};
int periods[3]={0,0,0},reorder = 0;
MPI_Comm comm;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Dims_create(size, ndims, dims);
if(rank == 0)
{
printf("\n=====REZULTATUL PROGRAMULUI '%s' \n",argv[0]);
for ( int i = 0; i < 3; i++ )
cout << "Numarul de procese pe axa "<< i<< " este "<< dims[i] << "; ";
cout << endl;
}
MPI_Barrier(MPI_COMM_WORLD);
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder, &comm);
MPI_Cart_coords(comm, rank, ndims, coords);
sleep(rank);
cout << "Procesul cu rankul " << rank << " are coordonatele (" << coords[0] << "," << coords[1] << "," <<
      coords[2] <<")"<< endl;
MPI_Barrier(MPI_COMM_WORLD);
if(rank == test_rank)
{
printf("Sunt procesul cu rankul %d (%d,%d,%d) vecinii mei sunt: \n", rank,
    coords[0],coords[1],coords[2]);
```

```
MPI_Cart_shift(comm,0,1,&left_x,&right_x);
if (left_x<0) {coords_left_x[0]=-1; coords_left_x[1]=-1;coords_left_x[2]=-1;}
else {MPI_Cart_coords(comm, left_x, ndims, coords_left_x); }
if (right_x<0) {coords_right_x[0]=1; coords_right_x[1]=1;coords_right_x[2]=1;}
else {MPI_Cart_coords(comm, right_x, ndims, coords_right_x); }
printf("   pe directia axei X : stanga %d(%d,%d,%d) dreapta %d(%d,%d,%d) \n",left_x,coords_left_x[0],
  coords_left_x[1],coords_left_x[2],    right_x ,coords_right_x[0], coords_right_x[1], coords_right_x[2]);
MPI_Cart_shift(comm,1,1,&up_y,&right_y);
if (up_y<0) {coords_left_y[0]=-1; coords_left_y[1]=-1;coords_left_y[2]=1;}
else {MPI_Cart_coords(comm, up_y, ndims, coords_left_y); }
if (right_y<0) {coords_right_y[0]=-1;coords_right_y[1]=-1;coords_right_y[2]=-1;}
else {MPI_Cart_coords(comm, right_y, ndims, coords_right_y); }
printf("   pe directia axei Y : stanga %d(%d,%d,%d) dreapta %d(%d,%d,%d)
  \n",up_y,coords_left_y[0],coords_left_y[1],coords_left_y[2],right_y,
  coords_right_y[0],coords_right_y[1], coords_right_y[2]);
MPI_Cart_shift(comm,2,1,&up_z,&down_z);
if (up_z<0) {coords_up_z[0]=-1; coords_up_z[1]=-1;coords_up_z[2]=-1;}
else {MPI_Cart_coords(comm, up_z, ndims, coords_up_z); }
if (down_z<0) {coords_down_z[0]=-1; coords_down_z[1]=-1; coords_down_z[2]=-1;}
else {MPI_Cart_coords(comm, down_z, ndims, coords_down_z); }
printf("   pe directia axei Z : jos %d(%d,%d,%d) sus %d(%d,%d,%d)
  \n",up_z,coords_up_z[0],coords_up_z[1],coords_up_z[2],down_z,coords_down_z[0],coords_down_z[1
  ],coords_down_z[2]);
 printf("Valorile negative semnifica lipsa procesului vecin!\n");
}
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
 [Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu_3_5_4.exe Exemplu_3_5_4.cpp
 [Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 6  -machinefile ~/nodes6 Exemplu_3_5_4.exe
=====REZULTATUL PROGRAMULUI 'Exemplu_3_5_4.exe'
Numarul de procese pe axa 0 este 3; Numarul de procese pe axa 1 este 2; Numarul de procese pe axa 2
este 1;
Procesul cu rankul 0 are coordonatele (0,0,0)
Procesul cu rankul 1 are coordonatele (0,1,0)
Procesul cu rankul 2 are coordonatele (1,0,0)
Procesul cu rankul 3 are coordonatele (1,1,0)
Procesul cu rankul 4 are coordonatele (2,0,0)
Procesul cu rankul 5 are coordonatele (2,1,0)
Sunt procesul cu rankul 2 (1,0,0) vecinii mei sunt:
   pe directia axei X : stanga 0(0,0,0) dreapta 4(2,0,0)
   pe directia axei Y : stanga -2(-1,-1,-1) dreapta 3(1,1,0)
   pe directia axei Z : jos -2(-1,-1,-1) sus -2(-1,-1,-1)
Valorile negative semnifica lipsa procesului vecin!
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 8  -machinefile ~/nodes6 Exemplu_3_5_4.exe
=====REZULTATUL PROGRAMULUI 'Exemplu_3_5_4.exe'
```

```
Numarul de procese pe axa 0 este 2; Numarul de procese pe axa 1 este 2; Numarul de procese pe axa 2 este 2;
Procesul cu rankul 0 are coordonatele (0,0,0)
Procesul cu rankul 1 are coordonatele (0,0,1)
Procesul cu rankul 2 are coordonatele (0,1,0)
Procesul cu rankul 3 are coordonatele (0,1,1)
Procesul cu rankul 4 are coordonatele (1,0,0)
Procesul cu rankul 5 are coordonatele (1,0,1)
Procesul cu rankul 6 are coordonatele (1,1,0)
Procesul cu rankul 7 are coordonatele (1,1,1)
Sunt procesul cu rankul 2 (0,1,0) vecinii mei sunt:
  pe directia axei X : stanga -2(-1,-1,-1) dreapta 6(1,1,0)
  pe directia axei Y : stanga 0(0,0,0) dreapta -2(-1,-1,-1)
  pe directia axei Z : jos -2(-1,-1,-1) sus 3(0,1,1)
Valorile negative semnifica lipsa procesului vecin!
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 27  -machinefile ~/nodes6 Exemplu_3_5_4.exe
=====REZULTATUL PROGRAMULUI 'Exemplu_3_5_4_new.exe'
Numarul de procese pe axa 0 este 3; Numarul de procese pe axa 1 este 3; Numarul de procese pe axa 2 este 3;
Procesul cu rankul 0 are coordonatele (0,0,0)
Procesul cu rankul 1 are coordonatele (0,0,1)
Procesul cu rankul 2 are coordonatele (0,0,2)
Procesul cu rankul 3 are coordonatele (0,1,0)
Procesul cu rankul 4 are coordonatele (0,1,1)
Procesul cu rankul 5 are coordonatele (0,1,2)
Procesul cu rankul 6 are coordonatele (0,2,0)
Procesul cu rankul 7 are coordonatele (0,2,1)
Procesul cu rankul 8 are coordonatele (0,2,2)
Procesul cu rankul 9 are coordonatele (1,0,0)
Procesul cu rankul 10 are coordonatele (1,0,1)
Procesul cu rankul 11 are coordonatele (1,0,2)
Procesul cu rankul 12 are coordonatele (1,1,0)
Procesul cu rankul 13 are coordonatele (1,1,1)
Procesul cu rankul 14 are coordonatele (1,1,2)
Procesul cu rankul 15 are coordonatele (1,2,0)
Procesul cu rankul 16 are coordonatele (1,2,1)
Procesul cu rankul 17 are coordonatele (1,2,2)
Procesul cu rankul 18 are coordonatele (2,0,0)
Procesul cu rankul 19 are coordonatele (2,0,1)
Procesul cu rankul 20 are coordonatele (2,0,2)
Procesul cu rankul 21 are coordonatele (2,1,0)
Procesul cu rankul 22 are coordonatele (2,1,1)
Procesul cu rankul 23 are coordonatele (2,1,2)
Procesul cu rankul 24 are coordonatele (2,2,0)
Procesul cu rankul 25 are coordonatele (2,2,1)
Procesul cu rankul 26 are coordonatele (2,2,2)
Sunt procesul cu rankul 2 (0,0,2) vecinii mei sunt:
  pe directia axei X : stanga -2(-1,-1,-1) dreapta 11(1,0,2)
  pe directia axei Y : stanga -2(-1,-1,-1) dreapta 5(0,1,2)
```

```
  pe directia axei Z : jos 1(0,0,1) sus -2(-1,-1,-1)
Valorile negative semnifica lipsa procesului vecin!
[Hancu_B_S@hpc Notate_Exemple]$
```

Exemplul 7.4a *Să se elaboreze un program MPI în limbajul C++ în care se creează un comunicator cu topologie cartezianã de tip paralelepiped şi să se determine pentru o faţetă a paralelepipedului procesele vecite (în direcţia acelor de cas) pentru oricare proces ce aparţine muchiilor aceste faţete.*

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 7.4a.

```
#include <mpi.h>
#include <stdio.h>
#include <iostream>
#include <vector>
using namespace std;
int main(int argc, char *argv[])
{
int rank,test_rank=2;
int size;
int ndims = 3;
int source, dest;
int left_y,right_y,right_x,left_x, up_z, down_z;
int dims[3]={0,0,0},coords[3]={0,0,0};
int coords_left_x[3]={0,0,0}, coords_right_x[3]={0,0,0}, coords_left_y[3]={0,0,0},
    coords_right_y[3]={0,0,0},   coords_up_z[3]={0,0,0}, coords_down_z[3]={0,0,0};
int periods[3]={0,0,0},reorder = 0;
MPI_Comm comm;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Dims_create(size, ndims, dims);
if(rank == 0)
{
printf("\n===== REZULTATUL PROGRAMULUI '%s' \n",argv[0]);
for ( int i = 0; i < 3; i++ )
{
cout << "Numarul de procese pe axa "<< i<< " este "<< dims[i] << "; ";
cout << endl;
}
cout << "===Rankul si coordonatele proceselor de pe fateta laterala a paralelepipedului pentru x= " <<
      dims[0]-1 << " ";
cout << endl;
}
MPI_Barrier(MPI_COMM_WORLD);
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder, &comm);
MPI_Cart_coords(comm, rank, ndims, coords);
sleep(rank);
```

```
if(coords[0] == dims[0]-1)
cout << "Procesul cu rankul " << rank << " are coordonatele (" << coords[0] << "," << coords[1] << "," <<
      coords[2] <<")"<< endl;
MPI_Barrier(MPI_COMM_WORLD);
if(rank == 0)
{
cout << "===Vecinii proceselor de pe muchiile fatetei laterali a paralelepipedului pentru x= "<< dims[0]-1
      << " ";
cout << endl;
}
sleep(rank);
if(coords[0] == dims[0]-1)
{
if (!((0<coords[1])&&(coords[1]<dims[1]-1)))
{
printf("Sunt procesul cu rankul %d (%d,%d,%d) vecinii mei sunt: \n",rank,coords[0],coords[1],coords[2]);
if (coords[2]=0)
MPI_Cart_shift(comm,1,1,&left_y,&right_y);
if (coords[2]=dims[2]-1)
MPI_Cart_shift(comm,1,-1,&left_y, &right_y);
if (left_y<0) {coords_left_y[0]=-1; coords_left_y[1]=-1;coords_left_y[2]=1;}
else {MPI_Cart_coords(comm, left_y, ndims, coords_left_y); }
if (right_y<0) {coords_right_y[0]=1;coords_right_y[1]=1;coords_right_y[2]=-1;}
else {MPI_Cart_coords(comm, right_y, ndims, coords_right_y); }
printf("   pe directia axei Y : stanga %d(%d,%d,%d) dreapta %d(%d,%d,%d)
      \n",left_y,coords_left_y[0],coords_left_y[1],coords_left_y[2],right_y,coords_right_y[0],coords_right
      _y[1],coords_right_y[2]);
if (coords[1]=0)  MPI_Cart_shift(comm,2,-1, &up_z, &down_z);
if (coords[1]=dims[1]-1)
MPI_Cart_shift(comm,2,1,&up_z,&down_z);
 if (up_z<0) {coords_up_z[0]=-1; coords_up_z[1]=-1;coords_up_z[2]=-1;}
else {MPI_Cart_coords(comm, up_z, ndims, coords_up_z); }
if (down_z<0) {coords_down_z[0]=-1; coords_down_z[1]=-1; coords_down_z[2]=-1;}
else {MPI_Cart_coords(comm, down_z, ndims, coords_down_z); }
printf("   pe directia axei Z : jos %d(%d,%d,%d) sus %d(%d,%d,%d)
      \n",up_z,coords_up_z[0],coords_up_z[1],coords_up_z[2],down_z,coords_down_z[0],coords_down_
      z[1],coords_down_z[2]);
}
}
MPI_Barrier(MPI_COMM_WORLD);
if(rank == 0)
printf("===Valorile negative semnifica lipsa procesului vecin!\n");
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc Notate_Exemple]$ /opt/openmpi/bin/mpiCC -o Exemplu_3_5_4a.exe
Exemplu_3_5_4a.cpp
```

```
[Hancu_B_S@hpc Notate_Exemple]$ /opt/openmpi/bin/mpirun -n 6  -machinefile ~/nodes6
Exemplu_3_5_4a.exe
===== REZULTATUL PROGRAMULUI 'Exemplu_3_5_4a.exe'
Numarul de procese pe axa 0 este 3;
Numarul de procese pe axa 1 este 2;
Numarul de procese pe axa 2 este 1;
===Rankul si coordonatele proceselor de pe fateta laterala a paralelepipedului pentru x= 2
Procesul cu rankul 4 are coordonatele (2,0,0)
===Vecinii proceselor de pe muchiile fatetei laterali a paralelepipedului pentru x= 2
Procesul cu rankul 5 are coordonatele (2,1,0)
Sunt procesul cu rankul 4 (2,0,0) vecinii mei sunt:
  pe directia axei Y : stanga 0(0,0,0) dreapta 0(0,0,0)
  pe directia axei Z : jos -2(-1,-1,-1) sus -2(-1,-1,-1)
Sunt procesul cu rankul 5 (2,1,0) vecinii mei sunt:
  pe directia axei Y : stanga 0(0,0,0) dreapta 0(0,0,0)
  pe directia axei Z : jos -2(-1,-1,-1) sus -2(-1,-1,-1)
===Valorile negative semnifica lipsa procesului vecin!

[Hancu_B_S@hpc Notate_Exemple]$ /opt/openmpi/bin/mpirun -n 8  -machinefile ~/nodes6
Exemplu_3_5_4a.exe
===== REZULTATUL PROGRAMULUI 'Exemplu_3_5_4a.exe'
Numarul de procese pe axa 0 este 2;
Numarul de procese pe axa 1 este 2;
Numarul de procese pe axa 2 este 2;
===Rankul si coordonatele proceselor de pe fateta laterala a paralelepipedului pentru x= 1
Procesul cu rankul 4 are coordonatele (1,0,0)
Procesul cu rankul 5 are coordonatele (1,0,1)
Procesul cu rankul 6 are coordonatele (1,1,0)
Procesul cu rankul 7 are coordonatele (1,1,1)
===Vecinii proceselor de pe muchiile fatetei laterali a paralelepipedului pentru x= 1
Sunt procesul cu rankul 4 (1,0,0) vecinii mei sunt:
  pe directia axei Y : stanga 6(1,1,0) dreapta -2(-1,-1,-1)
  pe directia axei Z : jos -2(-1,-1,-1) sus 5(1,0,1)
Sunt procesul cu rankul 5 (1,0,1) vecinii mei sunt:
  pe directia axei Y : stanga 7(1,1,1) dreapta -2(-1,-1,-1)
  pe directia axei Z : jos 4(1,0,0) sus -2(-1,-1,-1)
Sunt procesul cu rankul 6 (1,1,0) vecinii mei sunt:
  pe directia axei Y : stanga -2(-1,-1,-1) dreapta 4(1,0,0)
  pe directia axei Z : jos -2(-1,-1,-1) sus 7(1,1,1)
Sunt procesul cu rankul 7 (1,1,1) vecinii mei sunt:
  pe directia axei Y : stanga -2(-1,-1,-1) dreapta 5(1,0,1)
  pe directia axei Z : jos 6(1,1,0) sus -2(-1,-1,-1)
===Valorile negative semnifica lipsa procesului vecin!

[Hancu_B_S@hpc Notate_Exemple]$ /opt/openmpi/bin/mpirun -n 27  -machinefile ~/nodes6
Exemplu_3_5_4a.exe
===== REZULTATUL PROGRAMULUI 'Exemplu_3_5_4a.exe'
Numarul de procese pe axa 0 este 3;
Numarul de procese pe axa 1 este 3;
```

```
Numarul de procese pe axa 2 este 3;
===Rankul si coordonatele proceselor de pe fateta laterala a paralelepipedului pentru x= 2
Procesul cu rankul 18 are coordonatele (2,0,0)
Procesul cu rankul 19 are coordonatele (2,0,1)
Procesul cu rankul 20 are coordonatele (2,0,2)
Procesul cu rankul 21 are coordonatele (2,1,0)
Procesul cu rankul 22 are coordonatele (2,1,1)
Procesul cu rankul 23 are coordonatele (2,1,2)
Procesul cu rankul 24 are coordonatele (2,2,0)
Procesul cu rankul 25 are coordonatele (2,2,1)
Procesul cu rankul 26 are coordonatele (2,2,2)
===Vecinii proceselor de pe muchiile fatetei laterali a paralelepipedului pentru x= 2
Sunt procesul cu rankul 18 (2,0,0) vecinii mei sunt:
  pe directia axei Y : stanga 21(2,1,0) dreapta -2(-1,-1,-1)
  pe directia axei Z : jos -2(-1,-1,-1) sus 19(2,0,1)
Sunt procesul cu rankul 19 (2,0,1) vecinii mei sunt:
  pe directia axei Y : stanga 22(2,1,1) dreapta -2(-1,-1,-1)
  pe directia axei Z : jos 18(2,0,0) sus 20(2,0,2)
Sunt procesul cu rankul 20 (2,0,2) vecinii mei sunt:
  pe directia axei Y : stanga 23(2,1,2) dreapta -2(-1,-1,-1)
  pe directia axei Z : jos 19(2,0,1) sus -2(-1,-1,-1)
Sunt procesul cu rankul 21 (2,1,0) vecinii mei sunt:
  pe directia axei Y : stanga 24(2,2,0) dreapta 18(2,0,0)
  pe directia axei Z : jos -2(-1,-1,-1) sus 22(2,1,1)
Sunt procesul cu rankul 23 (2,1,2) vecinii mei sunt:
  pe directia axei Y : stanga 26(2,2,2) dreapta 20(2,0,2)
  pe directia axei Z : jos 22(2,1,1) sus -2(-1,-1,-1)
Sunt procesul cu rankul 24 (2,2,0) vecinii mei sunt:
  pe directia axei Y : stanga -2(-1,-1,-1) dreapta 21(2,1,0)
  pe directia axei Z : jos -2(-1,-1,-1) sus 25(2,2,1)
Sunt procesul cu rankul 25 (2,2,1) vecinii mei sunt:
  pe directia axei Y : stanga -2(-1,-1,-1) dreapta 22(2,1,1)
  pe directia axei Z : jos 24(2,2,0) sus 26(2,2,2)
Sunt procesul cu rankul 26 (2,2,2) vecinii mei sunt:
  pe directia axei Y : stanga -2(-1,-1,-1) dreapta 23(2,1,2)
  pe directia axei Z : jos 25(2,2,1) sus -2(-1,-1,-1)
===Valorile negative semnifica lipsa procesului vecin!
[Hancu_B_S@hpc Notate_Exemple]$
```