## Lectia 14. MODALITATI DE GESTIONARE A DATELOR ÎN OPENMP.

### *14.1 Clauze privind atributele de domeniu al datelor (Data Scope Attribute Clauses)*

O problemă importantă pentru programarea OpenMP este înţelegerea şi utilizarea domeniului acoperit de date. Deoarece OpenMP se bazează pe modelul de programare cu memorie partajată, cele mai multe variabile sunt utilizate în comun (shared) prin default.

Clauzele privind atributele de domeniu ale datelor sunt utilizate pentru a defini explicit cum trebuie utilizate variabilele în domenii. În lista clauzelor regăsim:

***private***
***firstprivate***
***lastprivate***
***shared***
***default***
***reduction***
***copyin***

Clauzele privind atributele de domeniu al datelor sunt utilizate în combinaţie cu mai multe dirctive (***parallel***, ***DO/for*** şi ***sections***) pentru a controla domeniul de utilizare a variabilelor incluse. Aceste clauze fac posibil controlul mediului de date în timpul executării constructorilor paraleli. Ele definesc cum şi care variabile din sectiunea secventială a programului sunt transferate către sectiunile paralel ale programului şi invers. Ele definesc care variabile vor fi vizibile tuturor firelor din sectiunile paralel şi care variabile vor fi alocate privat de toate firele.

*Notă:* clauzele privind atributele de domeniu au efect numai în extinderea lor lexicală/statică.

#### *14.1.1. Clauza PRIVATE*

*Scop:* Clauza **private** declară variabile care sunt private pentru fiecare fir.

*Formatul în C/C++:*

```
private (list)
```

*Note:* Variabilele **private** se comportă după cum urmează:

Un obiect nou de acelaşi tip se declară o dată pentru fiecare fir din fascicul. Toate referirile la obiectul originar sunt înlocuite cu referiri la obiectul nou. Variabilele declarate **private** sunt neinitializate pentru fiecare fir.

***Exemplu 14.1.1.*** *În acest exemlu se ilustrează modul de utilizare a variabilelor de tip* ***privat*** *.*

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
int main(int argc, char *argv[])
{
int n=1,iam;
printf("Valoarea lui n pana la directiva parallel cu clauza private: %d\n", n);
#pragma omp parallel private(n,iam)
    {
    sleep(omp_get_thread_num());
    printf("   OpenMP procesul %d-valoarea lui n dupa clauza private: %d\n",
    omp_get_thread_num(),n);
    n=omp_get_thread_num();
    printf("   OpenMP procesul %d-valoarea lui n dupa initializare de catre fir:
    %d\n",omp_get_thread_num(), n);
    }
```

```
printf("Valoarea lui n dupa directiva parallel cu clauza private: %d\n", n);
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.1.exe
    Exemplu2.1.1.cpp[1]
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu2.1.1.exe
Valoarea lui n pana la directiva parallel cu clauza private: 1
  OpenMP procesul 0-valoarea lui n dupa clauza private: 0
  OpenMP procesul 0-valoarea lui n dupa initializare de catre fir: 0
  OpenMP procesul 1-valoarea lui n dupa clauza private: -1
  OpenMP procesul 1-valoarea lui n dupa initializare de catre fir: 1
  OpenMP procesul 2-valoarea lui n dupa clauza private: 0
  OpenMP procesul 2-valoarea lui n dupa initializare de catre fir: 2
  OpenMP procesul 3-valoarea lui n dupa clauza private: 0
  OpenMP procesul 3-valoarea lui n dupa initializare de catre fir: 3
Valoarea lui n dupa directiva parallel cu clauza private: 1
[Hancu_B_S@hpc Open_MP]$
```

### *14.1.2. Clauza SHARED*

*Scop:* Clauza SARED declară în lista ei variabile care sunt partajate între toate firele fascicolului.

*Formatul în C/C++:*

```
shared (listă)
```

*Note*: O variabilă partajată există numai într-o locaţie de memorie şi toate firele pot citi sau scrie la acea adresă. Este în responsabilitatea programatorului a asigura că firele multiple au acces potrivit la variabilele **`shared`** (cum ar fi prin sectiunile **`critical`**).

***Exemplu 14.1.2.*** *În acest exemlu se ilustrează modul de utilizare a variabilelor de tip* ***`shared`***.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int i, m[5];
printf("Vectorul m pana la directiva parallel shared:\n");
/* Se initializeaza vectorul m */
for (i=0; i<5; i++){
m[i]=0;
printf("%d\n", m[i]);
}
#pragma omp parallel shared(m)
    {
    /* Se atribuie valoarea 1 elementului cu indicele egal cu numarul          firului din
    vectorul m */
    m[omp_get_thread_num()]=1;
    }
printf("Valoarea vectorului dupa  directiva parallel shared:\n");
for (i=0; i<5; i++) printf("%d\n", m[i]);
```

[1] Numele programului corespunde numelui exemplului din notele de curs Boris HÎNCU, Elena CALMÎŞ "MODELE DE PROGRAMARE PARALELĂ PE CLUSTERE. PARTEA II. Programare OpenMP şi mixtă MPI-OpenMP". Chisinau 2018.

```
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC  -fopenmp -o Exemplu2.1.2.exe
    Exemplu2.1.2.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -machinefile ~/nodes
    Exemplu2.1.2.exe
Vectorul m pana la directiva parallel shared:
0
0
0
0
0
Valoarea vectorului dupa  directiva parallel shared:
1
1
1
1
0
```

### *14.1.3. Clauza DEFAULT*

*Scop:* Clauza ***default*** permite utilizatorului să specifice prin default un domeniu ***private***, ***shared*** sau ***none*** pentru toate variabilele din extinderea lexicală a oricărei regiuni paralele.

*Formatul în C/C++:*

```
default (shared | none)
```

*Note:* Variabile specifice pot fi absolvite de default utilizând clauzele ***private***, ***shared***, ***firstprivate***, ***lastprivate*** şi ***reduction***. Specificatia în C/C++ din OpenMP nu include "***private***" ca un default posibil. Totusi, unele implementări pot avea prevăzută această opţiune.

*Restrictii:* numai clauza **default** poate fi specificată pe o directivă **parallel**.

### *14.1.4. Clauza FIRSTPRIVATE*

*Scop:* Clauza **firstprivate** combină comportarea clauzei **private** cu iniţializarea automată a variabilelor din lista ei.

*Formatul în C/C++:*

```
firstprivate (listă)
```

*Note:* Variabilele din listă sunt iniţializate potrivit cu valorile obiectelor lor originare înainte de intrarea în constructia paralelă sau de lucru partajat.

***Exemplu 14.1.3.*** *În acest exemlu se ilustrează modul de utilizare a variabilelor de tip* ***fistprivat***.

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
int main(int argc, char *argv[])
{
int n=1;
printf("Valoarea lui n pana la directiva parallel cu clauza firstprivate: %d\n", n);
#pragma omp parallel firstprivate(n)
{
sleep(omp_get_thread_num());
    printf("   OpenMP procesul %d-valoarea lui n dupa clauza firstprivate: %d\n",
    omp_get_thread_num(),n);
```

```
n=omp_get_thread_num();
printf("   OpenMP procesul %d-valoarea lui n dupa initializare de catre fire : %d\n",
     omp_get_thread_num(),n);
}
printf("Valoarea lui n dupa directiva parallel cu clauza firstprivate: %d\n", n);
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.3.exe
     Exemplu2.1.3.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
     0-1 Exemplu2.1.3.exe
Valoarea lui n pana la directiva parallel cu clauza firstprivate: 1
  OpenMP procesul 0-valoarea lui n dupa clauza firstprivate: 1
  OpenMP procesul 0-valoarea lui n dupa initializare de catre fire : 0
  OpenMP procesul 1-valoarea lui n dupa clauza firstprivate: 1
  OpenMP procesul 1-valoarea lui n dupa initializare de catre fire : 1
  OpenMP procesul 2-valoarea lui n dupa clauza firstprivate: 1
  OpenMP procesul 2-valoarea lui n dupa initializare de catre fire : 2
  OpenMP procesul 3-valoarea lui n dupa clauza firstprivate: 1
  OpenMP procesul 3-valoarea lui n dupa initializare de catre fire : 3
Valoarea lui n dupa directiva parallel cu clauza firstprivate: 1
[Hancu_B_S@hpc Open_MP]$
```

### *14.1.5. Clauza LASTPRIVATE*

*Scop:* Clauza **`lastprivate`** combină comportarea clauzei **`private`** cu copierea din ultima iteraţie din buclă sau secţiune în variabila obiect originară.

*Formatul în C/C++:*

```
lastprivate (listă)
```

*Note:*Valorile copiate înapoi în variabilele obiect originare se obţin din ultima iteraţie sau secţiune (secvenţială) a constructului care o conţine.

***Exemplu 14.1.4.*** *În acest exemlu se ilustrează modul de utilizare a variabilelor de tip* ***`lastprivat`***.

```
 #include <stdio.h>
#include <omp.h>
#include <iostream>
int main(int argc, char *argv[])
{
int n=0;
printf("Valoarea n în zona secventiala a programului (inainte de constructorul paralel):
     %d\n", n);
  #pragma omp parallel
  {
    #pragma omp sections lastprivate(n)
    {
     sleep(omp_get_thread_num());
     #pragma omp section
     {
printf("Valoarea n pentru firul %d (în #pragma omp section-pana la initializare):
     %d\n",omp_get_thread_num(), n);
     n=1;
printf("Valoarea n pentru firul %d (în #pragma omp section-dupa initializare): %d\n",
     omp_get_thread_num(), n);
```

```
      }
      #pragma omp section
      {
printf("Valoarea n pentru firul %d (în #pragma omp section- pana la initializare):
      %d\n",omp_get_thread_num(), n);
      n=2;
printf("Valoarea n pentru firul %d (în #pragma omp section- dupa initializare):
      %d\n",omp_get_thread_num(), n);
      }
      #pragma omp section
      {
printf("Valoarea n pentru firul %d (în #pragma omp section-pana la initializare):
      %d\n",omp_get_thread_num(), n);
      n=3;
printf("Valoarea n pentru firul %d (în #pragma omp section-dupa initializare):
      %d\n",omp_get_thread_num(), n);
       }
    }
sleep(omp_get_thread_num());
  printf("Valoarea n pentru firul %d: %d (dupa #pragma omp
     section)\n",omp_get_thread_num(), n);
  }
printf("Valoarea n în zona secventiala a programului(dupa constructorul paralel): %d\n", n);
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.4.exe
      Exemplu2.1.4.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu2.1.3.exe
Valoarea n în zona secventiala a programului (inainte de constructorul paralel): 0
Valoarea n pentru firul 1 (în #pragma omp section-pana la initializare): -1
Valoarea n pentru firul 1 (în #pragma omp section-dupa initializare): 1
Valoarea n pentru firul 3 (în #pragma omp section-pana la initializare): 0
Valoarea n pentru firul 3 (în #pragma omp section-dupa initializare): 3
Valoarea n pentru firul 0 (în #pragma omp section- pana la initializare): 0
Valoarea n pentru firul 0 (în #pragma omp section- dupa initializare): 2
Valoarea n pentru firul 0: 3 (dupa #pragma omp section)
Valoarea n pentru firul 1: 3 (dupa #pragma omp section)
Valoarea n pentru firul 2: 3 (dupa #pragma omp section)
Valoarea n pentru firul 3: 3 (dupa #pragma omp section)
Valoarea n în zona secventiala a programului(dupa constructorul paralel): 3
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu2.1.3.exe
Valoarea n în zona secventiala a programului (inainte de constructorul paralel): 0
Valoarea n pentru firul 3 (în #pragma omp section-pana la initializare): 0
Valoarea n pentru firul 3 (în #pragma omp section-dupa initializare): 1
Valoarea n pentru firul 0 (în #pragma omp section- pana la initializare): 0
Valoarea n pentru firul 0 (în #pragma omp section- dupa initializare): 2
Valoarea n pentru firul 3 (în #pragma omp section-pana la initializare): 1
Valoarea n pentru firul 3 (în #pragma omp section-dupa initializare): 3
Valoarea n pentru firul 0: 3 (dupa #pragma omp section)
Valoarea n pentru firul 1: 3 (dupa #pragma omp section)
Valoarea n pentru firul 2: 3 (dupa #pragma omp section)
```

```
Valoarea n pentru firul 3: 3 (dupa #pragma omp section)
Valoarea n în zona secventiala a programului(dupa constructorul paralel): 3
[Hancu_B_S@hpc Open_MP]$
```

Astfel, în regiune paralela (până la iniţializare) valoarea lun n nu este determinată, la ieşirea din regiunea paralelă valoarea lui n este egală cu ultima valoare initializată

### *14.1.6. Clauza COPYIN*

*Scop:* Clauza **copyin** asigură un mijloc de a atribui aceeaşi valoare variabilelor **threadprivate** pentru toate firele unui fascicul.

*Formatul în C/C++:*

```
copyin (listă)
```

*Note:* Lista conţine numele variabilelor de copiat. Variabilele firului master sunt sursa tuturor cópiilor. Firele fasciculului sunt iniţializate cu valorile lor la intrarea în constructul paralel.

***Exemplu 14.1.5.*** *În acest exemlu se ilustrează modul de utilizare a variabilelor de tip* ***copyin*** *.*

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
n=1;
#pragma omp parallel copyin(n)
{
sleep(omp_get_thread_num());
printf("Valoarea n în prima regiune paralela a firului %d: %d\n", omp_get_thread_num(),n);
}
printf("************************************\n");
  printf("Aici firul Master executa un cod serial\n");
  printf("************************************\n");
n=2;
#pragma omp parallel copyin(n)
{
sleep(omp_get_thread_num());
printf("Valoarea n în a doua regiune paralela a firului %d: %d\n",
     omp_get_thread_num(),n);
}
printf("************************************\n");
  printf("Aici firul Master executa un cod serial\n");
  printf("************************************\n");
#pragma omp parallel
{
sleep(omp_get_thread_num());
printf("Valoarea n în a treia regiune paralela a firului %d: %d\n", omp_get_thread_num(),n);
}
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.5.exe
     Exemplu2.1.5.cpp
```

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu2.1.4.exe
Valoarea n în prima regiune paralela a firului 0: 1
Valoarea n în prima regiune paralela a firului 1: 1
Valoarea n în prima regiune paralela a firului 2: 1
Valoarea n în prima regiune paralela a firului 3: 1
**********************************
Aici firul Master executa un cod serial
**********************************
Valoarea n în a doua regiune paralela a firului 0: 2
Valoarea n în a doua regiune paralela a firului 1: 2
Valoarea n în a doua regiune paralela a firului 2: 2
Valoarea n în a doua regiune paralela a firului 3: 2
**********************************
Aici firul Master executa un cod serial
**********************************
Valoarea n în a treia regiune paralela a firului 0: 2
Valoarea n în a treia regiune paralela a firului 1: 2
Valoarea n în a treia regiune paralela a firului 2: 2
Valoarea n în a treia regiune paralela a firului 3: 2
[Hancu_B_S@hpc Open_MP]$
```

### *14.1.6. Clauza REDUCTION*

*Scop*: Clauza **reduction** execută o operaţie de reducere pe variabilele care apar în listă. Pentru fiecare fir se crează o copie privată pentru fiecare variabilă din listă. La sfârşitul operaţiei de reducere, variabila de reducere este aplicată tuturor cópiilor private ale variabilelor partajate şi rezultatul final este scris în variabila globală folosită partajat.

*Formatul în C/C++:*

```
reduction (operator: listă)
```

*Restrictii:* Variabilele din listă trebuie să fie variabile scalare cu nume. Ele nu pot fi masive sau variabile de tipul stucturilor. Ele trebuie de asemenea să fie declarate **shared** în contextul care le include. Operaţiile de reducere pot să nu fie asociative pentru numere reale.

***Exemplu 14.1.6*** *În acest exemplu se ilusrează modul de utilizare a clauzei* **reduction**. *Se determina produsul scalar a doi vectori. Iteratiile buclei paralele vor fi distribuite în blocuri fiecărui fir din fascicol. La finalul constructiei buclei paralele toate firele vor aduna valorile "rezultatelor" lor pentru a actualiza copia globală din firul* **master.**

```
##include <stdio.h>
#include <omp.h>
#include <iostream>
main ()
{
int i, n, chunk,k;
float a[100], b[100], result;
/* Se initializeaza valorile */
n = 100;
chunk = 10;
result = 0.0;
for (i=0; i < n; i++)
{
a[i] = 1;//i * 1.0;
b[i] = 1;//i * 2.0;
```

```
}
omp_set_num_threads(2);
#pragma omp parallel default(shared) private(i,k) reduction(+:result)
      {
      k=0;
         #pragma omp for schedule(dynamic,chunk) nowait
         for (i=0; i < n; i++)
         {
         k=k+1;
         result = result + (a[i] * b[i]);
         }
         sleep(omp_get_thread_num());
         printf("Procesul OpenMP cu numarul %d, a determinat %d elemente ale produsului
      scalar egal cu %f\n", omp_get_thread_num(),k,result);

      }
printf("Produsul scalar este= %f\n",result);
}
```

Rezultatele vor fi urmatoarele:

1) Cazul ...schedule(static,chunk)...

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.6.exe
      Exemplu2.1.6.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu2.1.5.exe
Procesul OpenMP cu numarul 0, a determinat 50 elemente ale produsului scalar egal cu
      50.000000
Procesul OpenMP cu numarul 1, a determinat 50 elemente ale produsului scalar egal cu
      50.000000
Produsul scalar este= 100.000000
[Hancu_B_S@hpc Open_MP]$
```

2) Cazul ...schedule(dynamic,chunk)...

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.6.exe
      Exemplu2.1.6.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu2.1.6.exe
Procesul OpenMP cu numarul 0, a determinat 90 elemente ale produsului scalar egal cu
      90.000000
Procesul OpenMP cu numarul 1, a determinat 10 elemente ale produsului scalar egal cu
      10.000000
Produsul scalar este= 100.000000
[Hancu_B_S@hpc Open_MP]$
```

Operatiile de reducere pot fi numai de urmatoarea forma:

**C/C++**

```
x = x op expr
x = expr op x (except subtraction)
x binop = expr
x++
++x
x--
--x
unde:
```

```
x este o variabilă scalară din listă
expr este o expresie scalară care nu face referire la
x
op nu este overloaded şi este +, *, -, /, &, ^, |, &&
sau ||
binop nu este overloaded şi este +, *, -, /, &, ^ sau
|
```

**Un sumar al clauzelor/directivelor OpenMP**

| Clauze | Directive | | | | | |
|---|---|---|---|---|---|---|
| | PARALLEL | DO/for | SECTIONS | SINGLE | PARALLEL DO/for | PARALLEL SECTIONS |
| **IF** | X | | | | X | X |
| **PRIVATE** | X | X | X | X | X | X |
| **SHARED** | X | X | | | X | X |
| **DEFAULT** | X | | | | X | X |
| **FIRSTPRIVATE** | X | X | X | X | X | X |
| **LASTPRIVATE** | X | X | X | | X | X |
| **REDUCTION** | X | X | X | | X | X |
| **COPYIN** | X | | | | X | X |
| **SCHEDULE** | | X | | | X | |
| **ORDERED** | | X | | | X | |
| **NOWAIT** | | X | X | X | | |

Următoarele directive OpenMP nu admit clauze:

**master**
**critical**
**barrier**
**atomic**
**flush**
**ordered**
**threadprivate**

Implememtările pot diferi şi diferă uneori de standard în ceea ce priveste acceptarea clauzelor de către fiecare directivă.

***Exemplu 14.1.7.*** *Să se calculeze valoarea aproximativă a lui* $\pi$ *prin integrare numerică cu formula* $\pi = \int_0^1 \frac{4}{1+x^2} dx$, *folosind formula dreptunghiurilor. Intervalul închis [0,1] se împarte într-un număr de n subintervale şi se însumează ariile dreptunghiurilor având ca bază fiecare subinterval.*

Mai jos este prezentat codul programuluiîn limbajul C++ în care se realizează cele menționate în exemplul 14.1.7[2].

```
#include <stdio.h>
#include <stdlib.h>
```

[2] Semnificaţia variabilelor MPIrank, Nodes va fi explicata în capitolul 4.

```
#include <iostream>
#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
 #define omp_get_thread_num() 0
#endif
double f(double y) {return(4.0/(1.0+y*y));}
int main()
{
double w, x, sum, pi,a,b;
int i,MPIrank;
int n = 1000000;
int Nodes=1;
MPIrank=0;
w = 1.0/n;
sum = 0.0;
a=(MPIrank+0.0)/Nodes;
b=(MPIrank+1.0)/Nodes;
omp_set_num_threads(2);
#pragma omp parallel private(x) shared(w,a,b) reduction(+:sum)
{
#pragma omp master
{
printf("Pentru fiecare proces MPI se genereaza %d procese OpenMP (fire)\n",
      omp_get_num_threads());
}
#pragma omp for nowait
for(i=0; i < n; i++)
{
x = a+(b-a)*w*(i-0.5);
sum = sum + f(x);
}
sleep(omp_get_thread_num());
printf("Procesul OpenMP cu numarul %d, a determinat integrala egala cu %f\n",
      omp_get_thread_num(),(b-a)*w*sum);
}
pi = (b-a)*w*sum;
printf("Valoare finala , pi = %f\n", pi);
}
```

Rezultatele executarii programului.

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu2.1.7.exe
      Exemplu2.1.7.cpp
 [Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu2.1.7.exe
Pentru fiecare proces MPI se genereaza 2 procese OpenMP (fire)
Procesul OpenMP cu numarul 0, a determinat integrala egala cu 1.854591
Procesul OpenMP cu numarul 1, a determinat integrala egala cu 1.287003
Valoare finala , pi = 3.141595
[Hancu_B_S@hpc Open_MP]$
```