

Care dintre urmatoarele variante care se completeaza in locul comentariului

//COD DE COMPLETAT

conduce la afisarea

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

atunci cand programul se executa cu 4 procese:

```
1. int main(int argc, char *argv[]) {
2.     int nprocs, myrank;
3.     int i;
4.     int *a, *b;
5.     MPI_Status status;
6.     MPI_Init(&argc, &argv);
7.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
8.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
9.     a = (int *) malloc( nprocs * sizeof(int));
10.    b = (int *) malloc( nprocs* nprocs * sizeof(int));
11.    for(int i=0;i<nprocs; i++) a[i]=nprocs*myrank+i;
12.    /*
13.    COD DE COMPLETAT
14.    */
15.    if (myrank ==0)
16.        for(i=0;i<nprocs*nprocs; i++) printf(" %d", b[i]);
17.    MPI_Finalize( );
18.    return 0;
19. }
20.
```

#### Varianta A

```
1. if (myrank>0)
2.     MPI_Send(a, nprocs, MPI_INT, 0, 10, MPI_COMM_WORLD);
3. else {
4.     for (i = 0; i < nprocs; i++) b[i] = a[i];
5.     for (i = 1; i < nprocs; i++)
6.         MPI_Recv(b + i * nprocs, nprocs, MPI_INT, i, 10, MPI_COMM_WORLD, &status);
7. }
```

#### Varianta B

```
1. for (i =0; i < nprocs; i++) b[i+nprocs*myrank] = a[i];
2. if (myrank>0) MPI_Recv(b , nprocs*(myrank+1), MPI_INT, (myrank-1), 10, MPI_COMM_WORLD, &status);
3. MPI_Send(b, nprocs*(myrank+1), MPI_INT, (myrank+1)%nprocs, 10, MPI_COMM_WORLD);
4. if (myrank==0) MPI_Recv(b , nprocs*nprocs, MPI_INT, (nprocs-1), 10, MPI_COMM_WORLD, &status)
```

#### Varianta C

1. MPI\_Gather(a, nprocs, MPI\_INT, b, nprocs, MPI\_INT, 0, MPI\_COMM\_WORLD);

☐ B

☒ C ✓

☐ A

The correct answers are:

A,

B,

C

Question **2**

Correct

Mark 1.00 out of 1.00

[Flag question](#)

Conform legii lui Amdahl acceleratia este limitata de procentul(fractia) partii secventiale(care nu poate fi paralelizata) a unui program. Daca pentru un caz concret avem procentul partii secventiale egal cu 25% cat este acceleratia maxima care se poate obtine (cf legii lui Amdahl)?

- ☐ 25
- ☒ 4 ✓
- ☐ 75
- ☐ 3

The correct answer is: 4

Question **3**

Correct

Mark 1.00 out of 1.00

 Flag question

Corespunzator clasificarii Flynn arhitecturile de tip cluster se incadreaza in clasa

- ☐ SISD
- ☒ MIMD ✓
- ☐ MISD
- ☐ SIMD

The correct answer is: MIMD

Question **4**

Correct

Mark 1.00 out of 1.00

🚩 Flag question

**Poate sa apara data-race la executia programului urmator?**

//

```
1. static int sum=0;
2. static const int MAX=10000;
3. void f1(int a[], int s, int e){
4.     for(int i=s; i<e; i++)    sum += a[i];
5. }
6. int main() {
7.     int a[MAX];
8.     thread t1(f1, ref(a), 0,  MAX/2);
9.     thread t2(f1, ref(a), MAX/2, MAX);
10.    t1.join(); t2.join();
11.    cout<<sum<<endl;
12.    return 0;
13. }
```

//


Select one:

- ☒ True ✓
- ☐ False

The correct answer is 'True'.

Question **5**

Incorrect


Mark 0.00 out of  
1.00 Flag question

Consideram executia urmatoarei program MPI cu 4 procese.

```
////////////////////////////////////
```

```
1. int main(int argc, char *argv[] ) {  
2.     int nprocs, myrank;  
3.     MPI_Status status;  
4.     MPI_Init(&argc, &argv);  
5.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);  
6.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);  
7.     int value = myrank*10;  
8.     int sum=0, tmp;  
9.     MPI_Send (&value, 1, MPI_INT, (myrank+1)%nprocs, 10, MPI_COMM_WORLD);  
10.    MPI_Recv( &tmp, 1, MPI_INT, (myrank-1+nprocs)% nprocs, 10, MPI_COMM_WORLD, &status);  
11.    sum+=tmp;  
12.    if (myrank ==0)  
13.        printf("%d", sum);  
14.    MPI_Finalize( );  
15.    return 0;  
16. }
```

Care dintre urmatoarele afirmatii sunt adevarate?

- ☐ executia produce deadlock pentru ca procesul de la care primeste procesul 0 nu este bine definit
- ☐ executia programului produce deadlock pentru ca nici un proces nu poate sa finalizeze comunicatia
- ☒ se executa corect si afiseaza 60 

The correct answer is:

executia programului produce deadlock pentru ca nici un proces nu poate sa finalizeze comunicatia

## Question 6

Correct

Mark 100 out of 100

Flag question

Se considera executia urmatoarelor programe MPI cu 4 procese. Care dintre rezultatele evidentiate sunt posibile?

```

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    printf("Hello world from processor with rank %d out of %d processors\n", world_rank, world_size);
    MPI_Finalize();
    printf("Good bye! ");
}

```

- ☒ Hello world from processor with rank 1 out of 4 processors✔

Good bye!

Hello world from processor with rank 0 out of 4 processors

Good bye!

Hello world from processor with rank 2 out of 4 processors

Good bye!

Hello world from processor with rank 3 out of 4 processors

Good bye!

- ☐ Hello world from processor with rank 4 out of 4 processors

Good bye!

Hello world from processor with rank 2 out of 4 processors

Good bye!

Hello world from processor with rank 3 out of 4 processors

Good bye!

Hello world from processor with rank 1 out of 4 processors

Good bye!

- ☒ Hello world from processor with rank 3 out of 4 processors✔

Good bye!

Hello world from processor with rank 2 out of 4 processors

Good bye!

Hello world from processor with rank 0 out of 4 processors

Good bye!

Hello world from processor with rank 1 out of 4 processors

Good bye!

- ☐ Hello world from processor with rank 1 out of 4 processors

Hello world from processor with rank 0 out of 4 processors

Hello world from processor with rank 3 out of 4 processors

Hello world from processor with rank 2 out of 4 processors

Good bye!

- ☐ Hello world from processor with rank 1 out of 4 processors

Hello world from processor with rank 2 out of 4 processors

Hello world from processor with rank 3 out of 4 processors

Hello world from processor with rank 4 out of 4 processors

Good bye!

## Ce se poate intampla la executia programului urmator?

//

```
1. public class Main {
2.     static Object l1 = new Object();
3.     static Object l2 = new Object();
4.     static int a = 2, b = 2;
5.
6.     public static void main(String args[]) throws Exception{
7.         T1 r1 = new T1();      T2 r2 = new T2();
8.         Runnable r3 = new T1(); Runnable r4 = new T2();
9.
10.        ExecutorService pool = Executors.newFixedThreadPool(4);
11.        pool.execute(r1); pool.execute(r2); pool.execute(r3); pool.execute(r4);
12.
13.        pool.shutdown();
14.        while ( !pool.awaitTermination(60,TimeUnit.SECONDS){ }
15.            System.out.println("a=" + a + "; b="+ b);
16.        }
17.
18.        private static class T1 extends Thread {
19.            public void run() {
20.                synchronized (l1) {
21.                    synchronized (l2) {
22.                        int temp = a;
23.                        a += b;
24.                        b += temp;
25.                    }
26.                }
27.            }
28.        }
29.
30.        private static class T2 extends Thread {
31.            public void run() {
32.                synchronized (l2) {
33.                    synchronized (l1) {
34.                        a--;
35.                        b--;
36.                    }
37.                }
38.            }
39.        }
40.    }
```

//

- ☒ poate aparea deadlock pentru ca obiectele l1 si l2 sunt blocate in ordine inversa. ✓
- ☒ rezultatul executiei este nedeterminist. ✓
- ☐ se afiseaza : a=-2; b=-2.
- ☐ se afiseaza : a=8; b=8.
- ☐ apare eroare pentru ca nu se accepta nested synchronized

## Question 8

Partially correct

Mark 0.67 out of 1.00

[Flag question](#)

## Ce se poate intampla la executia programului urmator?

```
1. public class Main {
2.     static int numar = 1;
3.
4.     public static void main(String args[]) throws Exception{
5.         ThrCall task1 = new ThrCall( 2 );
6.         ThrCall task2 = new ThrCall( 3 );
7.
8.         ExecutorService pool = Executors.newFixedThreadPool( 2 );
9.         Future<Integer> future1 = pool.submit( task1 );
10.        Future<Integer> future2 = pool.submit( task2 );
11.        pool.shutdown();
12.
13.        Integer result1 = future1.get();
14.        Integer result2 = future2.get();
15.        System.out.println( "rez1 = " + result1 + "; rez2 = " + result2 );
16.    }
17.
18.    static class ThrCall implements Callable<Integer> {
19.        int n;
20.        public ThrCall( int n ){
21.            this.n=n;
22.        }
23.
24.        @Override
25.        public Integer call() throws Exception {
26.            for (int i = 0; i < n; i++) {
27.                numar *= numar;
28.            }
29.            return numar;
30.        }
31.    }
32. }
```

- ////////////////////////////////////
- ☐ nu poate aparea "data-race" pentru ca metodele call() nu scriu aceeasi resursa.
  - ☐ poate aparea deadlock
  - ☒ poate afisa: rez1 = 1; rez2 = 1 ✓
  - ☐ nu poate aparea deadlock
  - ☒ poate aparea "data-race" pentru ca metodele call() scriu aceeasi resursa. ✓

The correct answers are: nu poate aparea deadlock,  
poate aparea "data-race" pentru ca metodele call() scriu aceeasi resursa,  
poate afisa: rez1 = 1; rez2 = 1



Question 9

Correct

Mark 1.00 out of 1.00

Flag question

Care varianta de definire pentru variabilele grid si block(de completat in locul comentariului) conduce la crearea unui numar de 1024 de threaduri CUDA pentru apelul functiei VecAdd? </p>

```
/**** definire grid si block - de completat
```

```
VecAdd<<< grid , block >>>(A, B, C);
```

- ☒ dim3 grid(4); dim3 block(16,16); ✓
- ☒ dim3 grid(4); dim3 block(256); ✓
- ☒ dim3 grid(8, 8); dim3 block(4, 4); ✓
- ☐ dim3 grid(8); dim3 block(256);

The correct answers are:

dim3 grid(4); dim3 block(16,16);,


dim3 grid(8, 8); dim3 block(4, 4);,

dim3 grid(4); dim3 block(256);


Question **10**

Incorrect

Mark 0.00 out of 1.00

 Flag question

Care dintre urmatoarele afirmatii sunt adevarate?

- ☒ scalabilitatea arhitecturilor cu memorie distribuita este mai mica decat cea a arhitecturilor cu memorie partajata 
- ☐ scalabilitatea arhitecturilor cu memorie distribuita este mai mare decat cea a arhitecturilor cu memorie partajata

The correct answer is: scalabilitatea arhitecturilor cu memorie distribuita este mai mare decat cea a arhitecturilor cu memorie partajata

Question **11**

Correct

Mark 1.00 out of 1.00

 Flag question

In cadrul implementarii folosind sablonul Client-Server cu varianta 'stateless server' sunt adevarate urmatoarele afirmatii:

- ☒ starea unei sesiuni (session state) este gestionata de catre client ✓
- ☐ starea unei sesiuni (session state) este gestionata de catre server
- ☒ securitatea poate fi afectata pentru ca informatia se transmite de fiecare data (la fiecare request) ✓

Your answer is correct.

The correct answers are:


starea unei sesiuni (session state) este gestionata de catre client,

securitatea poate fi afectata pentru ca informatia se transmite de fiecare data (la fiecare request)

Question **12**

Correct

Mark 1.00 out of 1.00

 Flag question

Se considera paralelizarea sortarii unui vector cu  $n=2^k$  ( $2$  la puterea  $k$ ) elemente prin metoda "merge-sort" folosind sablonul de programare paralela Divide&impera.

In conditiile in care avem un numar nelimitat de procesoare, se poate ajunge la un anumit moment al executie la un grad maxim de paralelizare egal cu

- ☐  $n / k$
- ☐  $n*k$
- ☒  $n/2$  ✓
- ☐  $k$

The correct answer is:  $n/2$

## Question 13

Incorrect


Mark 0.00 out of 1.00

[Flag question](#)

Se considera executia cu 4 procese a urmatoarei program MPI.

```
1. int main(int argc, char *argv[] ) {
2.     int nprocs, myrank, mpi_err;
3.     int chunk=4;
4.     int *a, *b;
5.     MPI_Init(&argc, &argv);
6.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
7.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
8.     if (myrank == 0) {
9.         a = new int[nprocs*chunk];
10.        for(int i=0;i<nprocs*chunk; i++) a[i]=1;
11.    }
12.    b = new int[chunk];
13.    MPI_Scatter(a, chunk, MPI_INT, b, chunk, MPI_INT, 0 ,MPI_COMM_WORLD);
14.    for(int i=1;i<chunk; i++) b[0]+=b[i];
15.    MPI_Gather(b, 1, MPI_INT, a, 1, MPI_INT, 0 ,MPI_COMM_WORLD);
16.    if( myrank == 0) {
17.        for(int i=0;i<nprocs; i++) printf ("%d ", a[i]);
18.    }
19.    MPI_Finalize( );
20.    return 0;
21. }
```

Care dintre urmatoarele variante pot fi rezultatul executiei?

- ☐ valori calculate gresit din cauza alocarii insuficiente a spatiului de memorie pentru tabloul b
- ☒ 1 1 1 1 
- ☐ 4 4 4 4
- ☐ executia nu se termina

The correct answer is:

4 4 4 4

Question **14**

Partially correct

Mark 0.50 out of 1.00

🚩 Flag question

Ce se poate intampla la executia programului urmator?

```
////////////////////////////////////
1. mutex myMutex1, myMutex2;
2. void foo1(int n) {
3.     myMutex1.lock();    myMutex2.lock();
4.     for (int i = 10 * (n - 1); i < 10 * n; i++) {
5.         cout << " " << i << " ";
6.     }
7.     myMutex1.unlock();    myMutex2.unlock();
8. }
9.
10. void foo2(int n) {
11.     myMutex2.lock();    myMutex1.lock();
12.     for (int i = 10 * (n - 1); i < 10 * n; i++) {
13.         cout << " " << i << " ";
14.     }
15.     myMutex2.unlock();    myMutex1.unlock();
16. }
17.
18. int main() {
19.     thread t1(foo1, 1);
20.     thread t2(foo2, 2);
21.     thread t3(foo1, 3);
22.     thread t4(foo2, 4);
23.     t1.join();    t2.join();    t3.join();    t4.join();
24.     return 0;
25. }
```


- ////////////////////////////////////
- ☐ Poate aparea deadlock
  - ☒ Afiseaza grupuri de cate 10 numere (0...9; 10...19; 20..29; 30..39); in interiorul grupului numerele sunt ordonate, iar afisarea grupurilor este aleatorie ✓
  - ☐ Afiseaza aleator numerele din intervalul [0, 39]
  - ☐ Nu poate aparea deadlock
  - ☐ Afiseaza in ordine numerele de la 0 la 39

The correct answers are:  
Poate aparea deadlock,  
Afiseaza grupuri de cate 10 numere (0...9; 10...19; 20..29; 30..39); in interiorul grupului numerele sunt ordonate, iar afisarea grupurilor este aleatorie

Question **15**

Correct

Mark 1.00 out of 1.00

 Flag question

Aceleratia teoretica unui program paralel se defineste folosind urmatoarea formula:

Se considera:

$T_s$  = Complexitatea-timp a variantei secventiale

$T_p$  = complexitatea-timp a variantei paralele

$p$  = numarul de procesoare folosite pentru varianta paralela.

- ☐  $p \cdot T_s / T_p$
- ☐  $T_p / T_s$
- ☒  $T_s / T_p$  ✓
- ☐  $T_s / (p \cdot T_p)$

The correct answer is:

$T_s / T_p$

Question **16**

Correct

Mark 1.00 out  
of 1.00

🚩 Flag question

Cate threaduri se folosesc la executia urmatorului kernel CUDA?

```
__global__ void VecAdd(float* A, float* B, float* C)
{
    ...
}

int main()
{
    int M= 8, N=256;
    ...
    VecAdd<<< M , N >>>(A, B, C);
    ...
}
```

☒ 2048 ✓

☐ 32

☐ 256

☐ 8

☐ 1024

The correct answer is:

2048



Question 17

Correct

Mark 1.00 out of 1.00

Flag question

```
1  #include <stdio.h>
2  #include "omp.h"
3
4  void main() {
5      int i, k, p, j;
6      int N=4;
7
8      int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
9      int B[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
10     int C[4][4] ;
11
12     omp_set_num_threads(16);
13
14     #pragma omp parallel for private(i,k) shared(A, B, C, N) schedule(static)
15     for (i = 0; i< N; i++) {
16         for (k=0; k< N; k++) {
17             C[i][k] = (A[i][k] + B[i][k]);
18         }
19     }
20 }
```

Care sunt variabilele shared, respectiv variabilele private:

- ☐ 1. Shared: C / private: A, B, i, k, N
- ☒ 2. Shared: A, B, C, N / private: i, k ✓
- ☐ 3. Shared: A, B, C / private: i, k, N

Your answer is correct.


The correct answer is:

Shared: A, B, C, N / private: i, k

Question **18**

Partially correct

Mark 0.67 out of 1.00

 Flag question

Overhead-ul in programele paralele se datoreaza:

- ☐ partitionarii dezechilibrate in taskuri
- ☒ timpului datorat interactiunilor interproces ✓
- ☒ timpului necesar distributiei de date per procese/threaduri ✓
- ☒ timpului de asteptare datorat sincronizarii ✓
- ☒ timpului necesar crearii threadurilor/proceselor ✓
- ☐ calcul in exces (repetat de fiecare proces/thread)

The correct answers are:

timpului datorat interactiunilor interproces,

timpului de asteptare datorat sincronizarii,

timpului necesar crearii threadurilor/proceselor,

timpului necesar distributiei de date per procese/threaduri, partitionarii dezechilibrate in taskuri ,

calcul in exces (repetat de fiecare proces/thread)

## Question 19

Correct

Mark 1.00 out of 1.00

Flag question

```
1  #include <stdio.h>
2  #include "omp.h"
3
4  void main() {
5      int i, k, p, j;
6      int N=4;
7
8      int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
9      int B[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
10     int C[4][4] ;
11
12     omp_set_num_threads(16);
13
14     #pragma omp parallel for private(i,k) shared(A, B, C, N) schedule(static) collapse(2)
15     for (i = 0; i< N; i++) {
16         for (k=0; k< N; k++) {
17             C[i][k] = (A[i][k] + B[i][k]);
18         }
19     }
20 }
```

La ce linie se creeaza/distrug thread-urile:

- ☐ Creează: 12, distrug 20
- ☒ Creează: 14, distrug 20 ✓
- ☐ Creează: 4, distrug 20
- ☐ Creează: 16, distrug 17

Your answer is correct.

The correct answer is:

Creează: 14, distrug 20

Question **20**

Correct

Mark 1.00 out of 1.00

🚩 Flag question

Cate thread-uri vor fi create (cu exceptia thr Main) si care este rezultatul afisat de programul de mai jos?

```
////////////////////////////////////
1. public class Main {
2.     public static void main(String[] args) throws InterruptedException {
3.         AtomicNr a = new AtomicNr(5);
4.
5.         for (int i = 0; i < 2; i++) {
6.             Thread t1 = new Thread()->{ a.Add(3); });
7.             Thread t2 = new Thread()->{ a.Add(2); });
8.             Thread t3 = new Thread()->{ a.Minus(1); });
9.             Thread t4 = new Thread()->{ a.Minus(1); });
10.
11.             t1.start(); t2.start(); t3.start(); t4.start();
12.             t1.join(); t2.join(); t3.join(); t4.join();
13.         }
14.         System.out.println("a = " + a);
15.     }
16. };
17.
18. class AtomicNr{
19.     private int nr;
20.     public AtomicNr(int nr){ this.nr = nr;}
21.
22.     public synchronized void Add(int nr) { this.nr += nr;}
23.     public synchronized void Minus(int nr){ this.nr -= nr;}
24.
25.     @Override
26.     public String toString() { return "" + this.nr;}
27. };
////////////////////////////////////
```

Cate thread-uri noi se creeaza si care este valoarea variabilei "a"?

- ☒ Nr threaduri: 8; a = 11. ✓
- ☐ Nr threaduri: 8; Valorile finale ale lui "a" pot fi diferite la fiecare rulare pentru ca programarea paralela este nedeterminista.
- ☐ Nr threaduri: 4; a = 11.
- ☐ Nr threaduri: 0; a = 11; pentru ca in acest caz avem un program secvential.
- ☐ Nr threaduri: 2; a = 5

The correct answer is: Nr threaduri: 8; a = 11.

Question **21**

Correct

Mark 1.00 out of 1.00

[Flag question](#)

Care dintre urmatoarele afirmatii sunt adevarate?

- ☒ o procedura a monitorului nu poate fi apelata simultan de catre 2 sau mai multe threaduri ✓
- ☐ un monitor este definit de un set de proceduri
- ☒ un monitor poate fi accesat doar prin procedurile sale ✓
- ☐ toate procedurile monitorului pot fi executate la un moment dat

The correct answers are: un monitor poate fi accesat doar prin procedurile sale, o procedura a monitorului nu poate fi apelata simultan de catre 2 sau mai multe threaduri

Question **22**  
Partially correct  
Mark 0.50 out of 1.00  
[Flag question](#)

Care dintre afirmatiile urmatoare sunt adevarate?

- ☐ Scalabilitatea unei aplicatii paralele este determinata de numarul de taskuri care se pot executa in paralel.
- ☒ Daca numarul de taskuri care se pot executa in paralel creste liniar odata cu cresterea dimensiunii problemei atunci aplicatia are scalabilitate buna. ✓
- ☐ Partionarea prin descompunere functionala conduce in general la aplicatii cu scalabilitate mai buna decat partitionarea prin descompunerea domeniului de date.

The correct answers are:  
Scalabilitatea unei aplicatii paralele este determinata de numarul de taskuri care se pot executa in paralel,  
Daca numarul de taskuri care se pot executa in paralel creste liniar odata cu cresterea dimensiunii problemei atunci aplicatia are scalabilitate buna.

Question **23**

Correct

Mark 1.00 out of 1.00

 Flag question

Pentru sablonul de proiectare paralela "Pipeline" sunt adevarate urmatoarele afirmatii:

- ☒ pentru a obtine o performanta cat mai buna este preferabil ca impartirea pe subtaskuri sa fie cat mai echilibrata ✓
- ☐ se poate obtine performanta prin paralelizare indiferent daca este nevoie de mai multe traversari ale pipeline-ului sau doar de o traversare
- ☐ pentru a avea o performanta cat mai buna este preferabil ca numarul de subtaskuri in care se descompune calculul sa fie cat mai mic
- ☒ calculul se imparte in mai multe subtask-uri care se pot executa de catre unitati de procesare diferite ✓

The correct answers are:  
calculul se imparte in mai multe subtask-uri care se pot executa de catre unitati de procesare diferite,  
pentru a obtine o performanta cat mai buna este preferabil ca impartirea pe subtaskuri sa fie cat mai echilibrata

Question **24**

Correct

Mark 1.00 out of 1.00

🚩 Flag question

**Care dintre urmatoarele afirmatii este adevarata:**

- ☐ 1. Daca sunt mai multe block-uri de tipul **section** decat thread-uri, exista riscul de a nu se procesa o parte dintre aceste block-uri.
- ☒ 2. Fiecare block de tipul **section** este executat de un thread. ✓
- ☐ 3. Ordinea executiei block-urilor de tipul **section** este determinata.

Your answer is correct.

The correct answer is:

Fiecare block de tipul **section** este executat de un thread.



## Question 25

Correct

Mark 1.00 out of 1.00

Flag question

```
1  #include <stdio.h>
2  #include "omp.h"
3
4  void main() {
5      int i, k, p, j;
6      int N=4;
7
8      int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
9      int B[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
10     int C[4][4] ;
11
12     omp_set_num_threads(16);
13
14     #pragma omp parallel for private(i, k, p, j) shared(A, B, C, N) schedule(dynamic)
15     for (p = 0; p < N * N; p++)
16         i = p / N;
17         k = p % N;
18
19         j = omp_get_thread_num();
20
21         C[i][k] = A[i][k] + B[i][k] * j;
22     }
23 }
```

Cate thread-uri se vor crea:

- ☐ 1. 7 + 1 main
- ☒ 2. 15 + 1 main ✓
- ☐ 3. Cate core-uri exista pe CPU
- ☐ 4. 16 + 1 main

Your answer is correct.

The correct answer is:

15 + 1 main


The correct answer is:  
NU

Question **27**

Incorrect

Mark 0.00 out  
of 1.00 Flag question**Ce rezultat poate produce executia cu 4 procese a urmatoarei program mpi ?**

```
1. int main(int argc, char* argv[]) {
2.     int nprocs, myrank;
3.     MPI_Status status;
4.     MPI_Init(&argc, &argv);
5.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
6.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
7.     int value = myrank * 10;
8.     int tmp=0;
9.     if (myrank == 0)
10.    {
11.        MPI_Send(&value, 1, MPI_INT, (myrank + 1) % nprocs, 10, MPI_COMM_WORLD);
12.        MPI_Recv(&tmp, 1, MPI_INT, (myrank - 1 + nprocs) % nprocs, 10, MPI_COMM_WORLD, &status);
13.    }
14.    else {
15.        MPI_Recv(&tmp, 1, MPI_INT, (myrank - 1 + nprocs) % nprocs, 10, MPI_COMM_WORLD, &status);
16.        value += tmp;
17.        MPI_Send(&value, 1, MPI_INT, (myrank + 1) % nprocs, 10, MPI_COMM_WORLD);
18.    }
19.    if (myrank == 0)
20.        printf("%d", tmp);
21.    MPI_Finalize();
22.    return 0;
23. }
```

☐ 60☐ 0☒ se poate produce deadlock 

Your answer is incorrect.

The correct answer is:

60

Question 28

Correct

Mark 1.00 out of 1.00

Flag question

Consideram urmatoarea schita de implementarea pentru un semafor:

```
count : INTEGER
blocked: CONTAINER
down
do
  if count > 0 then
    count := count - 1
  else
    blocked.add(P)      -- P is the current process
    P.state := blocked  -- block process P
  end
end
up
do
  if blocked.is_empty then
    count := count + 1
  else
    Q := blocked.remove -- select some process Q
    Q.state := ready    -- unblock process Q
  end
end
```

Daca CONTAINER este o structura de tip FIFO atunci care dintre urmatoarele afirmatii sunt adevarate ?

- ☐ aceasta varianta de implementare defineste un "weak-semaphor" (semafor slab)
- ☐ aceasta varianta de implementare nu este "starvation-free"
- ☒ aceasta varianta de implementare este "starvation-free" ✓
- ☒ aceasta varianta de implementare defineste un "strong-semaphor" (semafor puternic) ✓

The correct answers are:

aceasta varianta de implementare este "starvation-free",

aceasta varianta de implementare defineste un "strong-semaphor" (semafor puternic)

Se considera executia cu 4 procese a urmatoarei program MPI.

```

1. int main(int argc, char* argv[]) {
2.     int nprocs, myrank, mpi_err;
3.     int i=0, value = 0;
4.     int* a=NULL, * b=NULL;
5.     MPI_Init(&argc, &argv);
6.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
7.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
8.     if (myrank == 0) {
9.         a = new int[nprocs];
10.        for (int i = 0; i < nprocs; i++) a[i] = i;
11.    }
12.    b = new int[1];
13.    MPI_Scatter(a, 1, MPI_INT, b, 1, MPI_INT, 0, MPI_COMM_WORLD);
14.    b[0] += myrank;
15.    printf("process %d b[0]= %d\n", myrank, b[0]);
16.    MPI_Reduce(b, &value, 1, MPI_INT, MPI_PROD, 0, MPI_COMM_WORLD);
17.    if (myrank == 0) {
18.        printf("value = %d \n", value);
19.    }
20.    MPI_Finalize();
21.    return 0;
22. }

```


Care este rezultatul executiei?

- ☐ value = 24
- ☐ programul nu se termina
- ☒ process 3 b[0]= 4 ✓  
 process 1 b[0]= 2  
 process 0 b[0]= 1  
 value = 24  
 process 2 b[0]= 3
- ☐ process 3 b[0]= 1  
 process 1 b[0]= 1  
 process 0 b[0]= 1  
 process 2 b[0]= 1  
 value = 4
- ☐ process 3 b[0]= 4  
 process 1 b[0]= 2  
 value = 0  
 process 0 b[0]= 1  
 process 2 b[0]= 3
- ☐ process 3 b[0]= 4  
 process 1 b[0]= 2  
 process 0 b[0]= 1  
 value = 10  
 process 2 b[0]= 3

Question **30**

Correct

Mark 1.00 out of 1.00

 Flag question

Un program paralel este optim din punct de vedere al costului daca:

- ☐ eficienta inmultita cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential
- ☒ timpul paralel inmultit cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential ✓
- ☐ acceleratia inmultita cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential
- ☐ acceleratia impartita cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential
- ☐ timpul paralel este de acelasi ordin de marime cu timpul secvential

The correct answer is: timpul paralel inmultit cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential

Question **31**

Correct

Mark 1.00 out of 1.00

🚩 Flag question

Ce valori corespund evaluarii teoretice a complexitatii-timp, acceleratiei, eficientei si costului pentru un program care face suma a 1024 de numere folosind 1024 de procesoare si un calcul de tip arbore binar? (Se ignora timpul de creare procese, distributie date, comunicatie, iar timpul necesar operatiei de adunare se considera egal cu 1.)


- ☐ [ 1 , 102.4 , 10, 102.4 ]
- ☐ [ 1 , 1024 , 1, 1024 ]
- ☒ [ 10 , 102.4 , 0.1, 10240] ✓
- ☐ [ 10 , 102.4 , 10.24, 1024 ]

The correct answer is: [ 10 , 102.4 , 0.1, 10240]

Question **32**

Correct

Mark 1.00 out of 1.00

 Flag question

Arhitecturile UMA sunt caracterizate de:

- ☒ acelasi timp de acces pentru orice locatie de memorie ✓
- ☐ identificator unic pentru fiecare procesor


The correct answer is: acelasi timp de acces pentru orice locatie de memorie



Question **33**

Correct

Mark 1.00 out of 1.00

 Flag question

Care dintre urmatoarele afirmatii este adevarata?

- ☒ Granularitatea unei aplicatii paralele este determinata de numarul de taskuri rezultate prin descompunerea calculului. ✓
- ☒ Granularitatea unei aplicatii paralele este definita ca dimensiunea minima a unei unitati secventiale dintr-un program, exprimata in numar de instructiuni. ✓
- ☒ Granularitatea unei aplicatii paralele se poate aproxima ca fiind raportul din timpul total de calcul si timpul total de comunicare. ✓

The correct answers are:  
Granularitatea unei aplicatii paralele este definita ca dimensiunea minima a unei unitati secventiale dintr-un program, exprimata in numar de instructiuni.,  
Granularitatea unei aplicatii paralele este determinata de numarul de taskuri rezultate prin descompunerea calculului.,  
Granularitatea unei aplicatii paralele se poate aproxima ca fiind raportul din timpul total de calcul si timpul total de comunicare.

Question **34**

Correct

Mark 1.00 out of 1.00

Flag question

Consideram urmatorul program MPI care se executa cu 4 procese.

//

```
1. int main(int argc, char *argv[] ) {
2.     int nprocs, myrank, tag=10;
3.     const int MAX_MESSAGE_LENGTH=50;
4.     MPI_Status status;
5.     MPI_Init(&argc, &argv);
6.     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
7.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
8.     int *a = new int[1];
9.     int value=0;
10.    a[0]=myrank;
11.    if (myrank == 0)    MPI_Send(a, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);
12.    MPI_Recv(&value, 1, MPI_INT, (myrank-1+nprocs)%nprocs, tag, MPI_COMM_WORLD, &status);
13.    a[0]+=value;
14.    if (myrank != 0)    MPI_Send(a, 1, MPI_INT, (myrank+1)%nprocs, tag, MPI_COMM_WORLD);
15.    if (myrank == 0)    printf("%d\n",a[0]);
16.    MPI_Finalize( );
17.    return 0;
18. }
```

//

**Intre ce perechi de procese se realizeaza comunicatia si in ce ordine se realizeaza comunicatiile?**

- ☐ (1->2) urmata de (2->3) urmata de (3->0)
- ☐ (0->1) urmata de (1->2) urmata de (2->3)
- ☒ (0->1) urmata de (1->2) urmata de (2->3) urmata de (3->0) ✓
- ☐ (1->2) urmata de (2->3) urmata de (3->0) urmata de (0->1)

The correct answer is:

(0->1) urmata de (1->2) urmata de (2->3) urmata de (3->0)

## Question 35

Correct

Mark 1.00 out of 1.00

Flag question

```
1  #include <stdio.h>
2  #include "omp.h"
3
4  void main() {
5      int i, k, p, j;
6      int N=4;
7
8      int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
9      int B[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
10     int C[4][4] ;
11
12     omp_set_num_threads(16);
13
14     #pragma omp parallel for private(i,k) shared(A, B, C, N) schedule(static)
15     for (i = 0; i < N; i++) {
16         for (k=0; k < N; k++) {
17             C[i][k] = (A[i][k] + B[i][k]);
18         }
19     }
20 }
```

Care va fi schema de distribuire a iteratiilor intre thread-urile create:

1. Thread 0: i = 0, k = 0-4 ✓

Thread 1: i = 1, k = 0-4

Thread 2: i = 2, k = 0-4

Thread 3: i = 3, k = 0-4

Thread 4-15: standby

Question 36  
Correct  
Mark 1.00 out  
of 1.00  
Flag question

```
1 #include <stdio.h>
2 #include "omp.h"
3
4 void main() {
5     int i, t, N = 15;
6     int a[N], b[N], c[N];
7
8     for (i=0; i < N; i++) a[i] = b[i] = 3;
9
10    omp_set_num_threads(3);
11
12    #pragma omp parallel shared(a,b,c) private(i, t) firstprivate(N)
13    {
14        #pragma omp single
15        t = omp_get_thread_num();
16
17        #pragma omp sections
18        {
19            #pragma omp section
20            {
21                for (i=0; i < N/3; i++)
22                    c[i] = a[i] / b[i] + t;
23            }
24            #pragma omp section
25            {
26                for (i=N/3; i < (N/3)*2; i++) {
27                    c[i] = a[i] + b[i] + t;
28                }
29            }
30            #pragma omp section
31            {
32                for (i=(N/3)*2; i < N; i++) {
33                    c[i] = a[i] * b[i] + t;
34                }
35            }
36        }
37    }
```

Care dintre urmatoarele afirmatii este adevarata:

- ☐ 1. **sections** este o directiva care nu determina executia in paralel a unui block de cod
- ☒ 2. Exista o bariera de sincronizare **implicita** la sfarsitul block-ului de tipul **sections**, astfel executia programului principal ramane suspendata pana cand toate thread-urile termina de procesat task-urile asociate in cadrul acestui block. ✓
- ☐ 3. nu este necesara gruparea codului in block-uri de tipul **section** pentru a indica ce dorim sa fie executat in paralel in cadrul unui block de tipul sections

Your answer is correct.

The correct answer is:

Exista o bariera de sincronizare **implicita** la sfarsitul block-ului de tipul **sections**, astfel executia programului principal ramane suspendata pana cand toate thread-urile termina de procesat task-urile asociate in cadrul acestui block.