Started on	Tuesday, 31 January 2023, 2:46 PM
State	Finished
Completed on	Tuesday, 31 January 2023, 3:31 PM
Time taken	45 mins 42 secs
Marks	25.90/36.00
Grade	**5.40** out of 7.50 (**71.94**%)

Question **1**

Partially correct

Mark 0.50 out of 1.00

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

- [x] 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 partially correct.

You have correctly selected 1.
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 **2**

Correct

Mark 1.00 out of 1.00

Ce rezultat poate produce executia cu 4 procese a urmatorului 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. }
```

- [ ] 0
- [ ] se poate produce deadlock
- [x] 60 ✔

Your answer is correct.

The correct answer is:
60

Question **3**

Correct

Mark 1.00 out of 1.00

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

- timpul paralel 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 ✔
- aceleratia impartita cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential
- eficienta inmultita cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential
- aceleratia inmultita cu numarul de procesoare 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 **4**

Partially correct

Mark 0.40 out of 1.00

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);**

- [ ] C
- [x] A ✔
- [ ] B

The correct answers are:

A,

B,

C

Question **5**

Correct

Mark 1.00 out of 1.00

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 nu este "starvation-free"
- [x] aceasta varianta de implementare defineste un "strong-semaphor" (semafor puternic) ✔
- [x] aceasta varianta de implementare este "starvation-free" ✔
- [ ] aceasta varianta de implementare defineste un "weak-semaphor" (semafor slab)

The correct answers are:
aceasta varianta de implementare este "starvation-free",

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

Question **6**

Partially correct

Mark 0.50 out of 1.00

Care dintre urmatoarele afirmatii sunt adevarate?

- [x] un monitor poate fi accesat doar prin procedurile sale ✔
- [x] o procedura a monitorului nu poate fi apelata simultan de catre 2 sau mai multe threaduri ✔
- [x] un monitor este definit de un set de proceduri ✖
- [ ] 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 **7**

Partially correct

Mark 0.33 out of 1.00

Overhead-ul in programele paralele se datoreaza:

- [x] timpului necesar crearii threadurlor/proceselor ✔
- [ ] calcul in exces (repetat de fiecare proces/thread)
- [ ] timpului datorat interactiunilor interproces
- [ ] timpului de asteptare datorat sincronizarii
- [ ] partitionarii dezechilibrate in taskuri
- [x] timpului necesar distributiei de date per procese/threaduri ✔

The correct answers are:
timpului datorat interactiunilor interproces,

timpului de asteptare datorat sincronizarii,

timpului necesar crearii threadurlor/proceselor,

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

calcul in exces (repetat de fiecare proces/thread)

Question **8**

Incorrect

Mark 0.00 out of 1.00

Ce se poate intampla la executia programului urmator?

//

```
1. mutex myMutex1, myMutex2;
2.  void foo1(int n) {
3.       myMutex1.lock();
4.       for (int i = 10 * (n - 1); i < 10 * n; i++) {
5.            cout << " " << i << " ";
6.       }
7.       myMutex1.unlock();
8. }
9. void foo2(int n) {
10.      myMutex2.lock();
11.      for (int i = 10 * (n - 1); i < 10 * n; i++) {
12.           cout << " " << i << " ";
13.      }
14.      myMutex2.unlock();
15. }
16.
17. int main()
18. {
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
- [x] 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 ✖
- [ ] Nu poate aparea deadlock
- [ ] Afiseaza aleator numerele din intervalul [0, 39]
- [ ] Afiseaza in ordine numerele de la 0 la 39

The correct answers are:
Nu poate aparea deadlock,

Afiseaza aleator numerele din intervalul [0, 39]

Question **9**

Incorrect

Mark 0.00 out of 1.00

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 , 1024 , 1, 1024] ✖
- [10 , 102.4 , 10.24, 1024]
- [1 , 102.4 , 10, 102.4]
- [10 , 102.4 , 0.1, 10240]

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

Question **10**

Correct

Mark 1.00 out of 1.00

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 ✔
- 8
- 256
- 1024
- 32

The correct answer is:
2048

Question **11**

Correct

Mark 1.00 out of 1.00

Cate thread-uri vor fi create (ce 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 < 5; 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 void Add(int nr) { this.nr += nr;}
23.     public void Minus(int nr){ this.nr -= nr;}
24.
25.     @Override
26.     public String toString() { return "" + this.nr;}
27. };
```

//

- [x] Nr threaduri: 20; Valorile finale ale lui "a" pot fi diferite la fiecare rulare din cauza "data race" ✔
- [ ] Nr threaduri: 20; a = 15
- [ ] Nr threaduri: 5; a = 5
- [ ] Nr threaduri: 0; a = 20
- [ ] Nr threaduri: 20; a = 5

The correct answer is: Nr threaduri: 20; Valorile finale ale lui "a" pot fi diferite la fiecare rulare din cauza "data race"

Question **12**

Correct

Mark 1.00 out of 1.00

Care din variantele de mai jos au acelasi efect cu apelul functiei **omp_set_num_threads(14)**:

1. **export OMP_NUM_THREADS=13**
2. **omp_get_num_threads()**
3. **num_threads(14)** ca si clauza intr-o directiva #pragma omp parallel ✔

Your answer is correct.

The correct answer is:
num_threads(14) ca si clauza intr-o directiva #pragma omp parallel

Question **13**

Correct

Mark 1.00 out of 1.00

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 **14**

Correct

Mark 1.00 out of 1.00

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(8, 8); dim3 block(4, 4); ✔
- dim3 grid(4); dim3 block(256); ✔
- 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 **15**

Correct

Mark 1.00 out of 1.00

Consideram executia urmatorului 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?

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

The correct answer is:
executia programului produce deadlock pentru ca nici un proces nu poate sa finalizeze comunicatia

Question **16**

Partially correct

Mark 0.50 out of 1.00

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(1);
11.         pool.execute(r1);   pool.execute(r2);  pool.execute(r3);   pool.execute(r4);
12.         pool.shutdown();
13.         while ( !pool.awaitTermination(60,TimeUnit.SECONDS)){ }
14.
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 (l1) {
33.                 synchronized (l2) {
34.                     a--;
35.                     b--;
36.                 }
37.             }
38.         }
39.     }
40. }
```

//

- [ ] se afiseaza : a=9; b=9
- [ ] se afiseaza : a=6; b=6
- [ ] se afiseaza : a=4; b=4.
- [x] se afiseaza : a=5; b=5.
- [ ] nu poate aparea deadlock pentru ca programul se executa secvential.

The correct answers are:
se afiseaza : a=5; b=5.,

nu poate aparea deadlock pentru ca programul se executa secvential.

Question **17**

Correct

Mark 1.00 out of 1.00

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)?

- 75
- 25
- 3
- 4 ✔

The correct answer is: 4

Question **18**

Correct

Mark 1.00 out of 1.00

Corespunzator clasificarii Flynn arhitecturile de tip cluster se incadreaza in clasa

- SIMD
- SISD
- MIMD ✔
- MISD

The correct answer is: MIMD

Question **19**

Correct

Mark 1.00 out of 1.00

```
#include <stdio.h>
#include "omp.h"

void  main() {
    int i, k, p, j;
    int N=4;

    int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int B[4][4] =  { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int C[4][4] ;

    omp_set_num_threads(16);

    #pragma omp parallel for private(i,k) shared(A, B, C, N) schedule(static)
    for (i = 0; i< N; i++) {
        for (k=0; k< N; k++) {
            C[i][k] = (A[i][k] +  B[i][k]);
        }
    }
}
```

Apelul **pragma omp parallel for** din exemplul de mai sus paralelizeaza executia ambelor structuri for ?

1. Fals ✔
2. Adevarat
3. Depinde de versiunea compilatorului folosita

Your answer is correct.

The correct answer is:
Fals

Question **20**

Correct

Mark 1.00 out of 1.00

Ce se poate intampla la executia programului urmator?

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 1. public class Main {
 2.     static int numar = 2;
 3.
 4.     public static void main(String args[]) throws Exception{
 5.         ThrCall task1 = new ThrCall(numar,2);
 6.         ThrCall task2 = new ThrCall(numar,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.
19. class ThrCall implements Callable<Integer> {
20.     int a, b;
21.     public ThrCall(int a, int b){
22.         this.a=a;
23.         this.b=b;
24.     }
25.
26.     @Override
27.     public Integer call() throws Exception {
28.         int p = 1;
29.         for (int i = 0; i < b; i++) {
30.             p *= a;
31.         }
32.         return p;
33.     }
34. }

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

- [x] nu poate sa apara deadlock. ✔
- [ ] poate aparea “data-race” pentru ca metodele call() scriu acceasi resursa.
- [ ] poate sa apara deadlock.
- [x] afiseaza: rez1 = 4; rez2 = 8. ✔
- [x] nu poate aparea “data-race” pentru ca metodele call() nu scriu acceasi resursa. ✔

The correct answers are: nu poate sa apara deadlock.,
nu poate aparea “data-race” pentru ca metodele call() nu scriu acceasi resursa.,
afiseaza: rez1 = 4; rez2 = 8.

Question **21**

Correct

Mark 1.00 out of 1.00

```c
#include <stdio.h>
#include "omp.h"

void  main() {
    int i, t, N = 15;
    int a[N], b[N], c[N];

    for (i=0; i < N; i++) a[i] = b[i] = 5;

    omp_set_num_threads(3);

    #pragma omp parallel shared(a,b,c) private(i, t) firstprivate(N)
        #pragma omp single
            t = omp_get_thread_num();

        #pragma omp sections
        {
            #pragma omp section
            {
                for (i=0; i < N/3; i++)
                    c[i] = a[i] / b[i] + t;
            }
            #pragma omp section
            {
                for (i=N/3; i < (N/3)*2; i++) {
                    c[i] = a[i] + b[i] + t;
                }
            }
            #pragma omp section
            {
                for (i=(N/3)*2; i < N; i++) {
                    c[i] = a[i] * b[i] + t;
                }
            }
        }
}
```

La ce linie se creeaza/distrug thread-urile:

- 1. Creeaza: 12, distrug: 36 ✔
- 2. Creeaza: 10, distrug: 36
- 3. Creeaza: 4, distrug 34
- 4. Creează: 16, distrug 36

Your answer is correct.

The correct answer is:
Creeaza: 12, distrug: 36

Question **22**

Correct

Mark 1.00 out of 1.00

In programul de mai jos poate aparea deadlock?

//

```
1. public class Main {
2.     int ID; int valThread;
3.
4.     public Main(int a, int b) {
5.         ID = a;
6.         valThread = b;
7.     }
8.
9.     public synchronized void  add(Main x)  {
10.        synchronized(x) {
11.            int temp = valThread;
12.            valThread =  x.valThread;
13.            x.valThread =  temp;
14.        }
15.    }
16.
17.    public static void main(String args[]) throws InterruptedException{
18.        Main obj1 = new Main(3,7);
19.        Main obj2 = new Main(2,9);
20.
21.        Thread threadObj1 = new Thread(()->{obj1.add(obj2);}); threadObj1.start();
22.        Thread threadObj2 = new Thread(()->{obj2.add(obj2);}); threadObj2.start();
23.        Thread threadObj3 = new Thread(()->{obj2.add(obj1);}); threadObj3.start();
24.
25.        threadObj1.join(); threadObj2.join(); threadObj3.join();
26.
27.        System.out.println(obj1.valThread + ", " + obj2.valThread);
28.    }
29. }
```

//

- (●) DA ✔
- ( ) NU

The correct answer is:
DA

Question **23**

Incorrect

Mark 0.00 out of 1.00

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
- k ✖
- n / k
- n/2

The correct answer is: n/2

Question **24**

Incorrect

Mark 0.00 out of 1.00

Se considera executia cu 4 procese a urmatorului 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] = 1;
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[i]);
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?

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

The correct answer is:
process 3 b[0]= 4

process 1 b[0]= 2

process 0 b[0]= 1

value = 24

process 2 b[0]= 3

Question **25**

Correct

Mark 1.00 out of 1.00

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?

- [ ] (0->1) urmata de (1->2) urmata de (2->3)
- [x] (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)
- [ ] (1->2) urmata de (2->3) urmata de (3->0)

The correct answer is:
(0->1) urmata de (1->2) urmata de (2->3) urmata de (3->0)

Question **26**

Correct

Mark 1.00 out of 1.00

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 **27**

Incorrect

Mark 0.00 out of 1.00

```
#include <stdio.h>
#include "omp.h"

void  main() {
    int i, k, p, j;
    int N=4;

    int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int B[4][4] =  { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int C[4][4] ;

    omp_set_num_threads(16);

    #pragma omp parallel for private(i,k) shared(A, B, C, N) schedule(static)
    for (i = 0; i< N; i++) {
        for (k=0; k< N; k++) {
            C[i][k] = (A[i][k] +  B[i][k]);
        }
    }
}
```

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

1. **Thread 0**: i = 0, k = 0 ✖

   Thread 1: i = 0, k = 1

   Thread 2: i = 0, k = 2

   Thread 3: i = 0, k = 3

   Thread 4: i = 1, k = 0

   Thread 5: i = 1, k = 1

   Thread 6: i = 1, k = 2

   Thread 7: i = 1, k = 3

   ....

   Thread 14: i = 3, k = 2

   Thread 15: i = 3, k = 3

2. **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

3. Ordinea de procesare nu este determinista, astfel ca fiecare thread va prelua in mod aleator task-urile care la randul lor vor avea dimensiune definita.

Your answer is incorrect.

The correct answer is:
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 **28**

Partially correct

Mark 0.67 out of 1.00

Care dintre urmatoarele afirmatii este adevarata?

- [x] Granularitatea unei aplicatii paralele este definita ca dimensiunea minima a unei unitati secventiale dintr-un program, exprimata ın numar de instructiuni. ✔
- [ ] Granularitatea unei aplicatii paralele este determinata de numarul de taskuri rezultate prin descompunerea calculului.
- [x] 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 ın 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 **29**

Correct

Mark 1.00 out of 1.00

Aceleratia teoretica unui program paralel se defineste folosind urmatoarea formula:

Se considera:

Ts = Complexitatea-timp a variantei secventiale

Tp= complexitatea-timp a variantei paralele

p=numarul de procesoare folosite pentru varianta paralela.

- ( ) p*Ts/Tp
- ( ) Ts/(p*Tp)
- ( ) Tp/Ts
- (•) Ts/Tp ✔

The correct answer is:
Ts/Tp

Question **30**

Partially correct

Mark 0.50 out of 1.00

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

- [ ] 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
- [ ] pentru a obtine o performanta cat mai buna este preferabil ca impartirea pe subtaskuri sa fie cat mai echilibrata
- [x] 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 **31**

Partially correct

Mark 0.50 out of 1.00

Care dintre afirmatiile urmatoare sunt adevarate?

- [x] Daca numarul de taskuri care se pot executa in paralel creste liniar odata cu cresterea dimensiunii problemei atunci aplicatia are scalabilitate buna. ✔
- [ ] Scalabilitatea unei aplicatii paralele este determinata de numarul de taskuri care se pot executa in paralel.
- [ ] 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 **32**

Correct

Mark 1.00 out of 1.00

```c
#include <stdio.h>
#include "omp.h"

void  main() {
    int i, k, p, j;
    int N=4;

    int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int B[4][4] =  { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int C[4][4] ;

    omp_set_num_threads(16);

    #pragma omp parallel for private(i, k, p, j) shared(A, B, C, N) schedule(dynamic)
        for (p = 0; p < N * N; p++)
            i = p / N;
            k = p % N;

            j = omp_get_thread_num();

            C[i][k] = A[i][k] + B[i][k] * j;
        }
}
```

Cate thread-uri se vor crea:

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

Your answer is correct.

The correct answer is:
15 + 1 main

Question **33**

Correct

Mark 1.00 out of 1.00

```c
#include <stdio.h>
#include "omp.h"

void  main() {
    int i, k, p, j;
    int N=4;

    int A[4][4] = { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int B[4][4] =  { {1, 2, 3, 4},{ 5, 6, 7, 8}, {9, 10, 11, 12} };
    int C[4][4] ;

    omp_set_num_threads(16);

    #pragma omp parallel for private(i, k, p, j) shared(A, B, C, N) schedule(dynamic)
        for (p = 0; p < N * N; p++)
            i = p / N;
            k = p % N;

            j = omp_get_thread_num();

            C[i][k] = A[i][k] + B[i][k] * j;
        }
}
```

Care sunt variabilele shared, respectiv variabilele private:

- 1. Shared: A, B, C, N / private: i, k, p, j ✔
- 2. Shared: A, B, C, N / private: k, p
- 3. Shared: A, B, C / private: i, k, p, j, N

Your answer is correct.

The correct answer is:
Shared: A, B, C, N / private: i, k, p, j

Question **34**

Incorrect

Mark 0.00 out of 1.00

Se considera executia cu 4 procese a urmatorului 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?

- [ ] executia nu se termina
- [ ] 4 4 4 4
- [ ] valori calculate gresit din cauza alocarii insuficiente a spatiului de memorie pentru tabloul b
- [x] 1 1 1 1 ✖

The correct answer is:
4 4 4 4

Question **35**

Correct

Mark 1.00 out of 1.00

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 **36**

Correct

Mark 1.00 out of 1.00

Se considera executia urmatorului program 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! ");
}
```

- [x] 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 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!

- [ ] 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!

- [x] 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!

The correct answers are:

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 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!

◄ Announcements

Jump to...