

1.

```
#include <iostream>
#include <mpi.h>
#define MAX 20

int nprocs, myrank;
double a[MAX], b[MAX], c[MAX];
MPI_Status status;
//
//init MPI

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    int value = myrank + 10;
    int sum = 0;
    MPI_Recv(&sum, 1, MPI_INT, (myrank - 1 + nprocs) / nprocs, 10,
MPI_COMM_WORLD, &status);
    sum += value;
    MPI_Send(&sum, 1, MPI_INT, (myrank + 1) % nprocs, 10, MPI_COMM_WORLD);
    if (myrank == 0)
        printf("%d", sum);
    MPI_Finalize();
    return 0;
}
```

Care din următoarele afirmatii sunt adevarate?

Se executa corect si afisaza 30.

Executia programului produce deadlock pentru ca procesul de la care primescte procesul 0 nu este bine definit.

Executia programului produce deadlock pentru ca nici un proces nu poate sa trimit inainte se primeasca.

2.Care dintre urmatoarele afirmatii sunt adevarate ?

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
cout:=count+1
else
Q:=blocked.remove--select some process Q
Qstate:=ready -- unblock process Q
end
end

```

aceasta varianta de implementare defineste un strong semaphore

aceasta varianta de implementare defineste un weak semaphore

aceasta varianta de implementare nu este "starvation free"

aceasta varianta de implementare este "starvation free"

```

3
int main(int argc, char* argv[])
{
    int nprocs, myrank;

    MPI_Init(&argc, &argv);
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    printf("Salutari de la %d",myrank);

    MPI_Finalize();
    printf("Program finalizat cu succes!");
}

```

```
        return 0;  
    }
```

Select one or more:

1.

Salutari de la 0

Salutari de la 1

Salutari de la 2

programul finalizat cu succes

programul finalizat cu succes

programul finalizat cu succes

2.

Salutari de la 3

Salutari de la 1

Salutari de la 2

programul finalizat cu succes

3.

Salutari de la 2

Salutari de la 0

Salutari de la 1

programul finalizat cu succes

programul finalizat cu succes

programul finalizat cu succes

4.

Salutari de la 0

Salutari de la 1

Salutari de la 2

programul finalizat cu succes

5.

Salutari de la 3

Salutari de la 1

Salutari de la 2

programul finalizat cu succes

programul finalizat cu succes

programul finalizat cu succes

6.

Salutari de la 2

Salutari de la 0

Salutari de la 1

programul finalizat cu succes

6.Ce se poate intampla la executia programului urmator?

```
public class Main {  
    static Object l1 = new Object();  
    static Object l2 = new Object();  
    static int a = 4, b = 4;
```

```
    public static void main(String args[]) throws Exception{  
        T1 r1 = new T1();    T2 r2 = new T2();  
        Runnable r3 = new T1(); Runnable r4 = new T2();  
        ExecutorService pool = Executors.newFixedThreadPool( 1 );  
        pool.execute( r1 ); pool.execute( r2 ); pool.execute( r3 ); pool.execute( r4 );  
        pool.shutdown();
```

```

while ( !pool.awaitTermination(60,TimeUnit.SECONDS)){ }

System.out.println("a=" + a + "; b="+ b);
}

private static class T1 extends Thread {
public void run() {
    synchronized (l1) {
        synchronized (l2) {
            int temp = a;
            a += b;
            b += temp;
        }
    }
}
}

private static class T2 extends Thread {
public void run() {
    synchronized (l2) {
        synchronized (l1) {
            a--;
            b--;
        }
    }
}
}
}

```

7. Select one or more:

1.

se afiseaza : a=9; b=9

2.

se afiseaza : a=13; b=13

3.

se afiseaza : a=12; b=12

4.

nu poate aparea deadlock

5.

se afiseaza : a=14; b=14

8.

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

```
////////////////////////////////////
public class Main {
    public static void main(String[] args) throws InterruptedException {
        AtomicNr a = new AtomicNr(5);

        for (int i = 0; i < 5; i++) {
            Thread t1 = new Thread(()->{ a.Add(3); });
            Thread t2 = new Thread(()->{ a.Add(2); });
            Thread t3 = new Thread(()->{ a.Minus(1); });
            Thread t4 = new Thread(()->{ a.Minus(1); });

            t1.start(); t2.start(); t3.start(); t4.start();
            t1.join(); t2.join(); t3.join(); t4.join();
        }
        System.out.println("a = " + a);
    }
};

class AtomicNr{
    private int nr;
    public AtomicNr(int nr){ this.nr = nr;}

    public void Add(int nr) { this.nr += nr;}
    public void Minus(int nr){ this.nr -= nr;}

    @Override
    public String toString() { return "" + this.nr;}
};
////////////////////////////////////
```

Select one or more:

1.

Nr threaduri: 5; a = 5

2.

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

3.

Nr threaduri: 0; a = 20

4.

Nr threaduri: 20; a = 15

5.

Nr threaduri: 20; a = 5

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

pentru a avea o performanta cat mai buna este preferabil ca numarul de subtaskuri in care se descompune calculul sa fie cat mai mic

2.

calculul se imparte in mai multe subtask-uri care se pot executa de catre unitati de procesare diferite

3.

se obtine performanta prin paralelizare daca este nevoie de mai multe traversari ale pipeline-ului

4.

pentru a obtine o performanta cat mai buna este preferabil ca impartirea pe subtaskurile sa fie cat mai echilibrata

Apare data-race la executia programului urmator?

```
////////////////////////////////////  
static int sum=0;  
static const int MAX=10000;  
void f1(int a[], int s, int e){  
    for(int i=s; i<e; i++) sum += a[i];  
}  
int main() {  
    int a[MAX];  
    thread t1(f1, ref(a), 0, MAX/2);  
    thread t2(f1, ref(a), MAX/2, MAX);
```



```

t1.join(); t2.join();
cout<<sum<<endl;
return 0;
}
////////////////////////////////////

```

TRUE - conform Fisier final PPD + Dinc

Care dintre urmatoarele afirmatii sunt adevarate?

Select one or more:

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

Conform Fisier final PPD + Dinc

Care este rezultatul executiei urmatorului program?

////////////////////////////////////

```

public class Main {
    static int value=0;
    static class MyThread extends Thread {
        Lock l; CyclicBarrier b;
        public MyThread(Lock l, CyclicBarrier b) {
            this.l = l; this.b = b;
        }
        public void run(){
            try{
                l.lock();
                value+=1;
                b.await();
            }
            catch (InterruptedException|BrokenBarrierException e) {
                e.printStackTrace();
            }
            finally { l.unlock();}
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Lock l = new ReentrantLock(); CyclicBarrier b = new CyclicBarrier(2);
        MyThread t1 = new MyThread( l, b ); MyThread t2 = new MyThread( l, b );
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.print(value);
    }
}

```

```
}  
////////////////////////////////////////////////////////////////
```

Select one or more:

- 1. se termina si afiseaza 1
- ☒ 2. nu se termina
- 3. se termina si afiseaza 2

conform Fisier final PPD + Dinc

Care dintre urmatoarele tipuri de comunicare MPI suspenda executia programului apelant pana cand comunicatia curenta este terminata?

Select one:

- 1. Asynchronous
- ☒ 2. Blocking
- 3. Nici una dintre variantele de mai sus
- 4. Nonblocking

conform Fisier final PPD + Dinc

Cate threaduri se folosesc la executia urmatorului kernel CUDA?

```
__global__ void VecAdd(float* A, float* B, float* C)  
{  
...  
}  
int main()  
{  
    int M= 16, N=8;  
    ...  
    VecAdd<<< M , N >>>(A, B, C);  
    ...  
}
```

Select one or more:

- ☒ 1. 128
- 2. 16
- 3. 8

conform Fisier final PPD + Dinc

Consideram executia urmatorului program MPI cu 3 procese.

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

```
int main(int argc, char *argv[]){  
    int nprocs, myrank;
```

```

    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    int value = myrank*10;
    int sum=0;
    MPI_Recv(&sum, 1, MPI_INT, (myrank-1+nprocs)/nprocs, 10, MPI_COMM_WORLD,
&status);
    sum+=value;
    MPI_Send(&sum, 1, MPI_INT, (myrank+1)%nprocs, 10, MPI_COMM_WORLD);
    if (myrank ==0)
        printf("%d", sum);
    MPI_Finalize( );
    return 0;
}
////////////////////

```

Care dintre urmatoarele afirmatii sunt adevarate?

Select one or more:

1. se executa corect si afiseaza 30
 - ☐ (?) 2. executia programului produce deadlock pentru ca procesul de la care primeste procesul 0 nu este bine definit
 - ☒ (X) 3. executia programului produce deadlock pentru ca nici un proces nu poate sa trimita inainte sa primeasca
- conform Fisier final PPD + Dino
- este posibil ca si 2 sa fie corect?

Consideram urmatorul program MPI care se executa cu 3 procese.

```

////////////////////
int main(int argc, char *argv[] ) {
    int nprocs, myrank;
    int i;
    int *a, *b;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    a = (int *) malloc( nprocs * sizeof(int));
    b = (int *) malloc( nprocs* nprocs * sizeof(int));
    for(int i=0;i<nprocs; i++) a[i]=nprocs*myrank+i;

    if (myrank>0) MPI_Recv(b, nprocs*(myrank+1), MPI_INT, (myrank-1), 10,
MPI_COMM_WORLD, &status);
    MPI_Send(b, nprocs*(myrank+1), MPI_INT, (myrank+1)%nprocs, 10, MPI_COMM_WORLD);
    if (myrank==0) MPI_Recv(b, nprocs*nprocs, MPI_INT, (myrank-1), 10, MPI_COMM_WORLD,
&status);
}

```

```

        MPI_Finalize( );
        return 0;
    }
    ///////////////////////////////////

```

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

Select one or more:

1. (0->1), (1-2), (2->0) in ordine aleatorie
2. (2->1), (1->0), (0->2) in ordine aleatorie
- ☒ 3. (0->1) urmata de (1-2) urmata de (2->0)
4. (2->1), urmata de (1->0), urmata de (0->2)
5. comunicatiile nu sunt bine definite pentru ca nu se realizeaza corect perechile (sender, receiver)

conform Fisier final PPD + Dinc

La ce linie se creeaza/distrug thread-urile:

```

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

void main(){
    int i,k;
    int N = 3;

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

    omp_set_num_threads(9);

    #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][j] = (A[i][k] + B[i][k]);
        }
    }
}

```

1. Creează: 17, distrug 18
2. Creează: 4, distrug 19
- ☒ 3. Creează: 14, distrug 20
4. Creează: 12, distrug 20

conform Document final PPD

La ce linie se creeaza/distrug thread-urile:

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

void main() {
    int i, t, N = 12;
    int a[N], b[N], c[N];

    for (i=0; i<N; i++) a[i] = b[i] = 3;

    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;
            }
        }
    }
}
```

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

(X)

conform Dino

Care sunt variabilele shared, respectiv variabilele private, in regiunea paralela:

1.

Shared: a, b, c, i, t

2.

Shared: a, b, c / private: i, N, t (conform excel doc)

3.

Shared: a, b, c / private: i, t

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

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

```
void main(){
```

```
    int i,k;
```

```
    int N = 3;
```

```
    int A[3][3] = {{1,2,3},{5,6,7},{8,9,10}};
```

```
    int B[3][3] = {{1,2,3},{5,6,7},{8,9,10}};
```

```
    int C[3][3] ;
```

```
    omp_set_num_threads(9);
```

```
    #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][j] = (A[i][k] + B[i][k]);
```

```
        }
```

```
    }
```

```
}
```

Care sunt variabilele shared, respectiv variabilele private:

(X) 1. Shared: A, B, C, N / private: i, k

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

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

conform conform Fisier final PPD + Dino

Cate thread-uri se vor crea:

Cate core-uri exista pe CPU

9 + 1 main



3 + 1 main

8 + 1 main (din excel fara cod?)

Se considera executia urmatoarei program MPI cu 2 procese

```
int main(int argc, char *argv[]){
    int nprocs, myrank;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    int value = myrank*10;
    if (myrank ==0) MPI_Recv( $value, 1, MPI_INIT, 0, 10, MPI_COMM_WORLD, &status);
    if (myrank ==1) MPI_Send( $value, 1, MPI_INIT, 0, 10, MPI_COMM_WORLD);
    if (myrank ==0) printf("%d", value);
    MPI_Finalize( );
    return 0;
}
```

Select one or more:

1. programul nu se termina pentru ca nu sunt bine definite comunicatiile
2. programul se termina si afiseaza valoarea 0

(X) 3. programul se termina si afiseaza valoarea 10

conform Dino

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, comunicare, iar timpul necesar operatiei de adunare se considera egal cu 1.)

Select one or more:

(X) 1. [1 , 1024 , 1, 1024]

2. [10 , 102.4 , 0.1, 10240]

(XX) 3. [1 , 102.4 , 10, 102.4]

4. [10 , 102.4 , 10.24, 1024]

(X) - conform Fisier final PPD

(XX) - conform Dino

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

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

```
void main() {
```

```
    int i,j,k,t;
```

```
    int N=4;
```

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

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

```
    int C[4][4] = ;
```

```
    omp_set_num_threads(3);
```

```
    #pragma omp parallel shared(A,B,C) private(i,j,k,t) firstprivate(N) {
```

```
        #pragma omp for schedule(dynamic)
```

```
        for (i=0; i<N; i=i+1){
```

```
            t = omp_get_thread_num();
```

```
            for (j=0; j<N; j=j+1) {
```

```
                C[i][j]=0.;
```

```
                for(k=0; k<N; k=k+1) {
```

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

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

Cate thread-uri se vor crea:

1. Cate core-uri exista pe CPU
☒ 2. 3 + 1 main
3. 15 + 1 main
4. 2 + 1 main
☒ - conform Dino



(A C E L E R A T I A am spus)

Acceleratia unui aplicatii paralele 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.

Select one or more:

- ☒ 1. T_s/T_p
☒ 2. $T_s/(p \cdot T_p)$
3. T_p/T_s
4. $p \cdot T_s/T_p$

☒ - conform Fisier final PPD

☒ - conform Dino

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.)

Select one or more:

1.
☒ [1 , 1024 , 1 , 1024] (fisier final)
2.
☒ [10 , 102.4 , 0.1 , 10240]
3.
☐ [1 , 102.4 , 10 , 102.4]
4.
☐ [10 , 102.4 , 10.24 , 1024]

```
#pragma omp parallel for  
for (i=1; i < 10; i++)  
{
```

```
    factorial[i] = i * factorial[i-1];  
}
```

Avem parte de data race in exemplul de mai sus ?

1. Fals, deoarece fiecarui thread ii vor fi asociate task-uri independente astfel incat nu este posibila o suprapunere in calcule.

2. Adevarat, pentru ca paralelizarea for este dinamica daca nu se specifica explicit

☒ 3. Adevarat, pentru ca exista posibilitatea ca un thread sa modifice valoarea factorial[i-1] in timp ce alt thread o foloseste pentru actualizarea elementului factorial[i]

conform Fisier final PPD + Dind

Corespunzator clasificarii Flynn arhitecturile de tip cluster se incadreaza in clasa

Select one or more:

1.

MISD

2.

SIMD

3.

☒ MIMD (conform Fisier final PPD)

4.

SISD

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

Select one or more:

1. timpul paralel este de acelasi ordin de marime cu timpul secvential

☒ 2. timpul paralel inmultit cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential

3. acceleratia inmultita cu numarul de procesoare este de acelasi ordin de marime cu timpul secvential

conform Fisier final PPD + Dind

Overhead-ul in programele paralele se datoreaza:

Select one or more:

☒ 1. timpului necesar crearii threadurilor/proceselor

☒ 2. timpului de asteptare datorat sincronizarii

☒ 3. partitionarii dezechilibrate in taskuri

4. interactiunii interproces

5. timpului necesar distributiei de date

6. calcul in exces (repetat de fiecare proces/thread)

conform Fisier final PPD + Dinc

Consideram urmatorul program MPI care trebuie completat in zona specificata de comentariul "COD de COMPLETAT".

Cu care dintre variantele specificate rezultatul executiei cu 3 procese va fi

0 1 2 3 4 5 6 7 8

```
int main(int argc, char argv[]) {
    int nprocs, myrank;
    int i;
    int *a, *b;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);


    a = (int *) malloc( nprocs * sizeof(int));
    b = (int *) malloc( nprocs* nprocs * sizeof(int));
    for(int i=0; i<nprocs; i++) a[i]=nprocs*myrank+i;

    /*
        COD de COMPLETAT
    */

    if (myrank==0)
        for(i=0; i<nprocs*nprocs; i++) printf(" %d", b[i]);

    MPI_Finalize( );
    return 0;
}
```

1.
MPI_Gather(a, nprocs, MPI_FLOAT, b, nprocs, MPI_FLOAT, 0, MPI_COMM_WORLD);

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

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

}

conform Fisier final PPD + Dino

Conform legii lui Amdahl, acceleratia este limitata de procentul(fractia) partii secventiale 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)?

Select one or more:

1. 75

2. 25

☒ 3. 4

conform Dino

```
#include "omp.h"
```

```
void main() {
```

```
    int i, t, N = 12;
```

```
    int a[N], b[N], c[N];
```

```
    for (i=0; i<N; i++) a[i] = b[i] = 3;
```

```
    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;
```

```
                }
```

```

    }
}

```

Are programul de mai sus o executie determinista ?

1. Nu, pentru ca nu vom obtine acelasi rezultat de ori cate ori am rula programul contand ordinea de executie a thread-urilor.

(X) 2. Da, pentru ca vom obtine acelasi rezultat de ori cate ori am rula programul chiar daca programul se va executa paralel.

3. Da, pentru ca block-urile de tipul section vor fi executate secvential si nu in paralel.

conform Dino

```

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

```

```

void main(){
    int i,k;
    int N = 3;

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

    omp_set_num_threads(9);

    #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][j] = (A[i][k] + B[i][k]);
        }
    }
}

```

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

(X) 1. Fals

(XX) 2. Adevarat

3. Depinde de versiunea compilatorului folosita

(X) - conform Fisier final PPD

(XX) - conform Dino

Ce se poate intampla la executia programului urmator?

```

////////////////////////////////////
public class Main {
    static Object l1 = new Object();
    static Object l2 = new Object();
    static int a = 4, b = 4;
}

```

```

public static void main(String args[]) throws Exception{
    T1 r1 = new T1();    T2 r2 = new T2();
    Runnable r3 = new T1(); Runnable r4 = new T2();
    ExecutorService pool = Executors.newFixedThreadPool( 1 );
    pool.execute( r1 );  pool.execute( r2 ); pool.execute( r3 );  pool.execute( r4 );
    pool.shutdown();
    while ( !pool.awaitTermination(60, TimeUnit.SECONDS)){ }

    System.out.println("a=" + a + "; b="+ b);
}

private static class T1 extends Thread {
    public void run() {
        synchronized (l1) {
            synchronized (l2) {
                int temp = a;
                a += b;
                b += temp;
            }
        }
    }
}

private static class T2 extends Thread {
    public void run() {
        synchronized (l2) {
            synchronized (l1) {
                a--;
                b--;
            }
        }
    }
}
}

```

//

Select one or more:

- ☒ 1. nu poate aparea deadlock
- ☐ 2. se afiseaza : a=14; b=14
- ☒ 3. se afiseaza : a=13; b=13
- ☐ 4. se afiseaza : a=9; b=9
- ☐ 5. se afiseaza : a=12; b=12

Conform Dint

Rezultatul executiei este nedeterminist (conform Fisier final PPD)

Care dintre afirmatiile urmatoare sunt adevarate?

Select one or more:

1. Partionarea prin descompunere functionala conduce in general la aplicatii cu scalabilitate mai buna decat partitionarea prin descompunerea domeniului de date.

2. Scalabilitatea unei aplicatii paralele este determinata de numarul de taskuri care se pot executa in paralel.

☒ 3. Daca numarul de taskuri care se pot executa in paralel creste liniar odata cu cresterea dimensiunii problemei atunci aplicatia are scalabilitate buna.

Conform Dino

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

☒ 1. pentru a avea o performanta cat mai buna este preferabil ca numarul de subtaskuri in care se descompune calculul sa fie cat mai mic

☒☒ 2. calculul se imparte in mai multe subtask-uri care se pot executa de catre unitati de procesare diferite

☒☒ 3. se obtine performanta prin paralelizare daca este nevoie de mai multe traversari ale pipeline-ului

☒ 4. pentru a obtine o performanta cat mai buna este preferabil ca impartirea pe subtaskurile sa fie cat mai echilibrata

(X) - conform Fisier final PPD

(XX) - conform Dino

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

```
////////////////////////////////////  
public class Main {  
    public static void main(String[] args) throws InterruptedException {  
        AtomicNr a = new AtomicNr(5);  
  
        for (int i = 0; i < 5; i++) {  
            Thread t1 = new Thread(()->{ a.Add(3); });  
            Thread t2 = new Thread(()->{ a.Add(2); });  
            Thread t3 = new Thread(()->{ a.Minus(1); });  
            Thread t4 = new Thread(()->{ a.Minus(1); });  
  
            t1.start(); t2.start(); t3.start(); t4.start();  
            t1.join(); t2.join(); t3.join(); t4.join();  
        }  
        System.out.println("a = " + a);  
    }  
};  
  
class AtomicNr{  
    private int nr;  
    public AtomicNr(int nr){ this.nr = nr;}  
  
    public void Add(int nr) { this.nr += nr;}
```

```

    public void Minus(int nr){ this.nr -= nr;}

    @Override
    public String toString() { return "" + this.nr;}
};
/////////////////////////////////////////////////////////////////

```

Select one or more:

- 1. Nr threaduri: 5; a = 5
- ☒ 2. Nr threaduri: 20; Valorile finale ale lui "a" pot fi diferite la fiecare rulare din cauza "data race"
- 3. Nr threaduri: 0; a = 20
- 4. Nr threaduri: 20; a = 15
- 5. Nr threaduri: 20; a = 5

Conform Fisier final PPD + Dinc

Apare data-race la executia programului urmator?

```

/////////////////////////////////////////////////////////////////

```

```

public class Test {
static int value=0;
static class MyThread extends Thread{
public void run() { value++; }
}
public static void main(String[] args) throws InterruptedException {
    MyThread t1 = new MyThread(); MyThread t2 = new MyThread();
    t1.start(); t2.start();
    t1.join(); t2.join();
    System.out.print(value);
}
}

```

- a) DA (conform Fisier final PPD)**
- b) NU

Consideram urmatoarea schita de implementarea pentru un semafor:

```

count: INTEGER
blocked: CONTAINER
down
do
    if count > 0 then
        count := count - 1
    else
        blocked.app(P)        -- P is current process

```



```

                P.state := blocked      -- blocked process P
            end
        end
    up
    do
        if blocked.is_empty then
            count := count + 1
        else
            Q := blocked.remove      -- selected some process Q
            Q.state := ready-- unblock process Q
        end
    end
end

```

Care dintre urmatoarele afirmatii sunt adevarate ?

Select one or more:

1. aceasta varianta de implementare defineste un "strong-semaphor" (semafor puternic)
- ☒ 2. aceasta varianta de implementare defineste un "weak-semaphor" (semafor slab)
- ☒ 3. aceasta varianta de implementare nu este "starvation-free"
4. aceasta varianta de implementare este "starvation-free"

Conform Dino

Se considera paralelizarea sortarii unui vector cu n elemente prin metoda "merge-sort" folosind sablonul 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

Select one or more:

1. log n
- ☒ 2. n / log n
- ☒ 3. n

☒ Conform Dino

☒ Conform Fisier final PPD

Arhitecturile UMA sunt caracterizate de:

Select one or more:

1. identificator unic pentru fiecare procesor
- ☒ 2. acelasi timp de acces pentru orice locatie de memorie

Conform Fisier final PPD + Dino

(U R M A T O R U L U I am zis)

urmatorului program se va executa cu 3 procese. Ce valoare se va afisa?

////////////////////////////////////

```

int main(int argc, char *argv[] ) {
    int nprocs, myrank;
    int i, value=0;
    int *a, *b;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

```

```

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
    a = (int*) malloc(nprocs * sizeof(int));
    for(int i=0;i<nprocs; i++) a[i]=i+1;
}
b = (int *) malloc( sizeof(int));
MPI_Scatter(a, 1, MPI_INIT, b, 1, MPI_INT, 0 ,MPI_COMM_WORLD);
    b[0] += myrank;
MPI_Reduce(b, &value, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if( myrank == 0) {
    printf("value = \"%d \\n\", value); }
MPI_Finalize( );
return 0;
}

```

////////////////////////////////////

Select one or more:

☒ 1. 9

☒ 2. 6

3. 12

☒ Conform Dino

☒ Conform Fisier final PPD