

# ALGORITMI PARALELI ȘI DISTRIBUIȚI

## Tema #3 Calcule colaborative în sisteme distribuite

**Responsabili:** Elena Apostol, Radu-Ioan Ciobanu

Termen de predare: 15-01-2023 23:59 (soft), 17-01-2023 23:59 (hard)  
Ultima modificare: 22-12-2022 05:50

### Cuprins

<b>Cerință</b>	<b>2</b>
<b>Detalii</b>	<b>2</b>
Stabilirea topologiei . . . . .	2
Realizarea calculelor . . . . .	4
Tratarea defectelor pe canalul de comunicație . . . . .	4
Tratarea partiționărilor . . . . .	4
<b>Execuție</b>	<b>5</b>
<b>Notare</b>	<b>5</b>
<b>Testare</b>	<b>6</b>

## Cerință

Să se implementeze un program distribuit în MPI în care procesele sunt grupate într-o topologie formată din patru clustere, fiecare din acestea având câte un coordonator și câte un număr arbitrar de procese worker. Procesele worker dintr-un cluster pot să comunice doar cu coordonatorul lor, iar cei patru coordonatori pot să comunice între ei într-o topologie de tip inel pentru a conecta clusterelor.

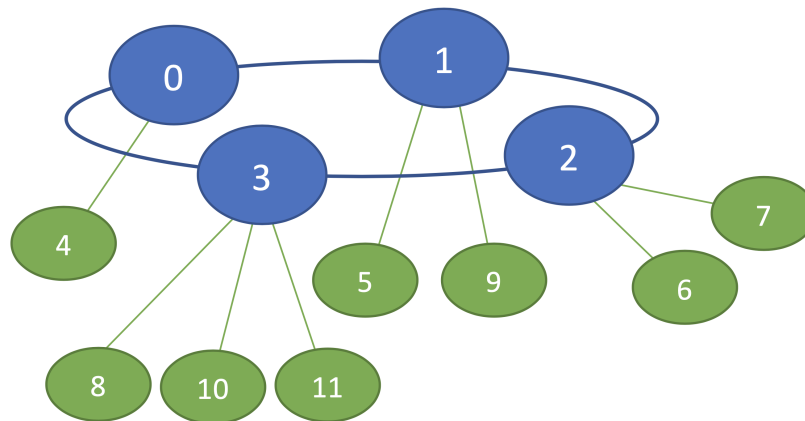
Scopul temei este ca toate procesele worker să ajungă să lucreze împreună, cu ajutorul coordonatorilor, pentru rezolvarea unor task-uri computaționale. Acest lucru se va realiza prin stabilirea topologiei și diseminarea ei către toate procesele, și apoi prin împărțirea calculelor în mod cât mai echilibrat între workeri.

## Detalii

Această temă este formată din patru etape, care se punctează individual (ultima din ele fiind bonus).

### Stabilirea topologiei

Sistemul distribuit implementat în această temă este format din patru clustere cu câte un număr arbitrar de procese worker, așa cum se poate observa în exemplul din figura de mai jos. Fiecare proces cunoaște faptul că sunt patru clustere încă de la începutul rulării programului.



Fiecare cluster are câte un coordonator, adică procesele cu rangurile 0, 1, 2 și 3 în cadrul implementării noastre (acest lucru fiind cunoscut din start de către toți coordonatorii). Fiecare coordonator este responsabil de propriile sale procese worker, așa cum se poate observa în figura de mai sus (unde fiecare linie dintre două procese reprezintă un canal de comunicație).

**Atenție!** Un proces nu poate să comunice decât pe baza legăturilor din topologie.

La începutul execuției programului distribuit, procesele coordonator vor citi informații despre procesele din clusterelor lor din patru fișiere de intrare (câte unul pentru fiecare coordonator), numite *cluster0.txt*, *cluster1.txt*, *cluster2.txt* și *cluster3.txt*. Pentru topologia prezentată în figura de mai sus, cele patru fișiere de intrare vor arăta astfel:

```
$ cat cluster0.txt
1
4

$ cat cluster1.txt
2
5
```

```
9
$ cat cluster2.txt
2
6
7

$ cat cluster3.txt
3
8
10
11
```

În exemplele de mai sus, prima linie reprezintă numărul de procese worker dintr-un cluster, iar următoarele linii conțin rangurile workerilor. Coordonatorul cu rangul 0 va citi din fișierul *cluster0.txt*, coordonatorul 1 va citi din *cluster1.txt*, coordonatorul 2 va citi din *cluster2.txt*, iar coordonatorul 3 va citi din *cluster3.txt*.

La începutul rulării programului distribuit, un proces worker nu știe ce rang are coordonatorul clusterului în care se află. De aceea, este sarcina coordonatorilor de cluster să își informeze workerii despre cine este coordonatorul lor.

Prima sarcină pe care trebuie să o rezolvați este **stabilirea topologiei sistemului distribuit în toate procesele**. La finalul acestei sarcini, trebuie ca fiecare proces (fie că e coordonator, fie că e worker) să cunoască topologia întregului sistem și să o afișeze pe ecran. Nu vi se cere să folosiți un algoritm anume pentru acest lucru, deci puteți să vă faceți propria logică, atât timp cât **procesele comunică doar pe baza legăturilor din topologie** (un worker poate comunica doar cu coordonatorul său, coordonatorii pot comunica între ei doar pe baza unei topologii de tip inel).

Fiecare mesaj direct trimis de un proces trebuie logat în terminal astfel (în exemplul de mai jos, procesul 0 a trimis un mesaj către procesul 4, iar procesul 3 a trimis un mesaj către procesul 11):

```
M(0,4)
M(3,11)
```

**Atenție!** Când afișați mesajele de mai sus, rangurile proceselor sunt cele din comunicatorul global.

În momentul în care un proces are topologia finală, va trebui să o afișeze în terminal astfel:

```
1 -> 0:4 1:5,9 2:6,7 3:8,10,11
```

În exemplul de mai sus, procesul 1 a aflat topologia și o afișează, iar topologia este cea din imaginea prezentată anterior. Așadar, cele patru clusterse sunt afișate în ordine, începând cu rangul coordonatorului, urmat de rangurile proceselor worker din acel cluster.

**Atenție!** Toate procesele trebuie să afișeze topologia atunci când o află.

Punctajul pe această etapă este primit în totalitate cât timp toate procesele ajung să afle și să afișeze topologia corectă fără a comunica direct cu procese cu care nu sunt conectate în topologie.

## Realizarea calculelor

Odată ce toate procesele cunosc topologia, urmează partea de calcule, care este coordonată de către procesul 0. Mai concret, procesul 0 va genera un vector  $V$  de dimensiune  $N$  (unde  $N$  este primit ca parametru la rularea programului și este inițial cunoscut doar de către procesul 0), unde  $V[K] = N - K - 1$  (cu  $K$  între 0 și  $N - 1$ ), care trebuie înmulțit cu 5 element cu element. Înmulțirea cu 5 a elementelor vectorului trebuie realizată **doar** de către procesele worker, deci este responsabilitatea proceselor coordonator să împartă calculele în mod cât mai echilibrat la procesele worker.

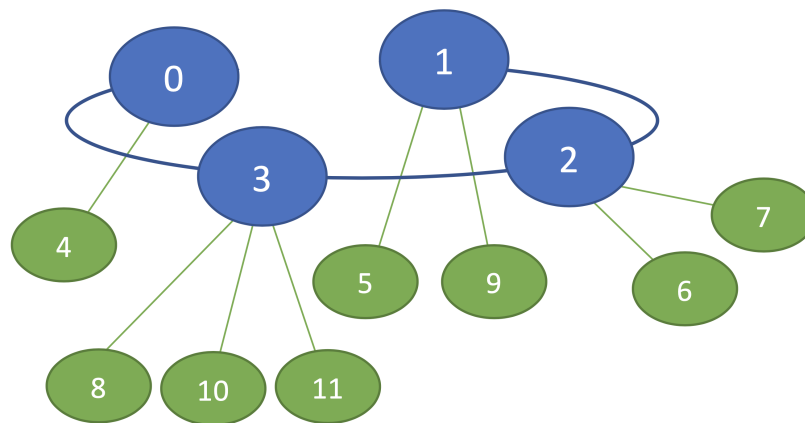
**Atenție!** Echilibrarea calculelor se face la nivel de procese worker, nu de cluster. Astfel, dacă avem de făcut 8000 de iterații în topologia de mai sus, fiecare worker va realiza câte 1000 (deci, în exemplul de topologie de mai sus, primul cluster va face 1000 de calcule, al doilea va face 2000, al treilea 2000, iar ultimul va face 3000).

Odată ce procesul 0 generează vectorul, va distribui mai departe calculele către clusterul său și către celelalte trei cluster, prin intermediul coordonatorilor. Când calculele sunt finalizate, vectorul rezultat trebuie reasamblat la procesul 0 și afișat de către acesta. Un exemplu de afișare poate fi observat mai jos, pentru un vector de dimensiune 12:

Rezultat: 55 50 45 40 35 30 25 20 15 10 5 0

## Tratarea defectelor pe canalul de comunicație

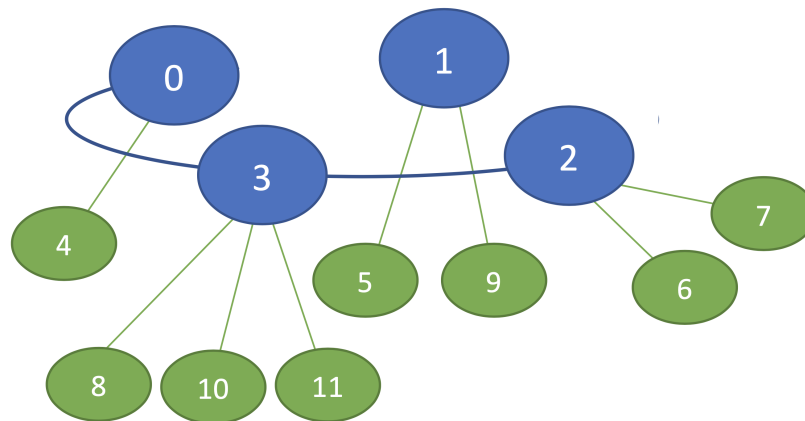
A treia parte a temei presupune **tratarea cazului când avem o eroare pe canalul de comunicație dintre procesele 0 și 1**. Mai precis, legătura dintre cele două procese dispare, așa cum se poate observa în imaginea de mai jos (unde avem aceeași structură a sistemului distribuit prezentată anterior).



Astfel, pentru această cerință, trebuie să faceți pașii de la primele două cerințe (stabilirea topologiei în toate procesele și apoi înmulțirea cu 5 a elementelor vectorului) în lipsa legăturii dintre procesele 0 și 1. Toate cerințele legate de logica de implementare și de afișări de la primele două puncte rămân în continuare valabile.

## Tratarea partiționărilor

A patra parte a temei este **bonus** și presupune **tratarea cazului când avem o partiționare în sistem**. Mai precis, procesul 1 ajunge să fie deconectat de procesele 0 și 2, deci el nu mai poate fi accesat din orice alt coordonator. Astfel, avem o partiționare a sistemului, ceea ce înseamnă că procesul 1 și workerii săi nu mai pot fi utilizați în realizarea calculelor, așa cum se poate observa în imaginea de mai jos (unde avem aceeași structură a sistemului distribuit prezentată anterior).



Astfel, pentru această cerință, trebuie să faceți pașii de la primele două cerințe (stabilirea topologiei în toate procesele și apoi înmulțirea cu 5 a elementelor vectorului) în condiții de partiționare. Toate cerințele legate de logica de implementare și de afișări de la primele două puncte rămân în continuare valabile.

**Atenție!** La stabilirea topologiei pentru această cerință, nu veți mai avea afișări identice în toate procesele, pentru că procesul 1 și workerii săi nu știu nimic despre celelalte procese și viceversa. În acest caz, un proces din partiția mare va afișa următorul mesaj pentru topologia din imaginea de mai sus:

```
4 -> 0:4 2:6,7 3:8,10,11
```

Un proces din partiția mică va afișa următorul mesaj:

```
5 -> 1:5,9
```

De asemenea, pentru că în acest caz procesul 0 nu poate ajunge deloc la procesul 1, numărul de procese care vor participa la calcule este mai mic, pentru că cele din partiția mică sunt excluse. Astfel, doar șase workeri vor realiza calculele, conform imaginii de mai sus.

## Execuție

Compilarea surselor voastre trebuie să rezulte într-un binar de MPI numit *tema3*, care se va rula în felul următor:

```
mpirun --oversubscribe -np <numar_procese> ./tema3 <dim_vec> <eroare_comunicatie>
```

Al doilea parametru dat la rulare va fi 0 dacă legătura dintre procesele 0 și 1 există (nu are loc o eroare pe canalul de comunicație, ceea ce înseamnă rularea pentru primele două cerințe), 1 dacă procesele 0 și 1 nu pot comunica direct (adică rularea pentru a treia cerință), sau 2 dacă are loc o partiție și coordonatorul 1 este izolat (adică rularea pentru cerința bonus).

**Atenție!** Se presupune că datele de intrare sunt întotdeauna corecte.

## Notare

Tema se va trimite și testa automat folosind VMChecker-Next prin încărcarea pe Moodle la [această adresă](#). Se va încărca o arhivă Zip care, pe lângă fișierele sursă, va trebui să conțină următoarele două fișiere în rădăcina arhivei:

- *Makefile* - cu directiva *build* care compilează tema voastră și generează un executabil numit *tema3* aflat în rădăcina arhivei
- *README* - fișier text în care să se descrie pe scurt implementarea temei.

Punctajul este divizat după cum urmează:

- **30p** - stabilirea corectă a topologiei în toate procesele atunci când nu sunt erori de comunicație (prima cerință)
- **30p** - corectitudinea calculelor realizate de procesele worker atunci când nu sunt erori de comunicație (a doua cerință)
- **20p** - realizarea corectă și echilibrată a calculelor atunci când legătura dintre procesele 0 și 1 dispare (a treia cerință)
- **20p** - claritatea codului și a explicațiilor din README
- **15p (bonus)** - realizarea corectă și echilibrată a calculelor atunci când are loc o partiție (a patra cerință).

Veți fi depunctați pentru următoarele elemente:

- **-100p** - un proces comunică direct cu un alt proces cu care nu este conectat în topologie (de exemplu, doi workeri comunică direct între ei, un worker dintr-un cluster comunică direct cu un coordonator al unui alt cluster, etc.)
- **-100p** - neutilizarea MPI
- **-100p** - citirea fișierelor de intrare de către alte procese decât coordonatorii de cluster
- **-30p** - neechilibrarea calculelor pe care procesele worker le realizează (această depunctare se va da în cazul în care există o diferență semnificativă între cantitățile de calcule pe care procesele worker trebuie să le facă)
- **-30p** - neafișarea la output a mesajelor de comunicare între procese
- **-30p** - afișarea la output de mesaje în plus față de cele specificate în enunț
- **-30p** - implicarea proceselor coordonator în realizarea calculelor.

## Testare

Pentru a vă putea testa tema și local, găsiți în [repository-ul temei](#) un set de fișiere de intrare de test, precum și un script Bash pe care îl puteți rula conform indicațiilor de la temele precedente (puteți consulta [acest document](#) pentru mai multe detalii).

La rulare, scriptul execută următorii pași:

1. compilează sursele
2. rulează trei teste unde există legătura dintre procesele 0 și 1
3. pentru fiecare test, se verifică dacă toate procesele afișează corect topologia, dacă rezultatul calculelor este corect, și dacă nu se trimit mesaje pe canale de comunicație inexistente
4. se rulează un test unde nu există conexiunea dintre procesele 0 și 1 (pentru a treia cerință) și se verifică dacă toate procesele afișează corect topologia, dacă rezultatul calculelor este corect, și dacă nu se trimit mesaje pe canale de comunicație inexistente

5. se rulează un test unde există o partiționare (pentru bonus) și se verifică dacă toate procesele afișează corect topologia, dacă rezultatul calculelor este corect, și dacă nu se trimit mesaje pe canale de comunicație inexistente
6. se calculează punctajul final din cele 80 de puncte (sau 95, cu bonus) alocate corectitudinii execuției (20 de puncte fiind rezervate pentru claritatea codului și a explicațiilor).

**Atenție!** Scriptul nu verifică toate potențialele depunctări (conform baremului), deci nota afișată de el nu va fi neapărat nota finală.