Proiectarea Sistemelor Paralele și Distribuite

Suport de Curs

16 iunie 2025

Cuprins

1 Concepte Fundamentale de Concurență și Procese **3**
- 1.1 Terminologie . . . 3
- 1.2 Procese . . . 3
- 1.3 Fire de Execuție (Threads) . . . 3
  - 1.3.1 Componente Thread . . . 3
  - 1.3.2 Fire de Execuție în C++ . . . 3
- 1.4 Partajarea Datelor și Condiții de Cursă . . . 4
  - 1.4.1 Variabile și Parametri . . . 4
  - 1.4.2 Condiții de Cursă (Race Conditions) . . . 4
- 1.5 Soluții pentru Condițiile de Cursă . . . 4
  - 1.5.1 Instrucțiuni Atomice . . . 4
  - 1.5.2 Mutexes (Mutual Exclusion) . . . 5
- 1.6 Blocaje (Deadlocks) . . . 5
- 1.7 Variabile Condiție (Condition Variables) . . . 5

2 Arhitecturi Paralele **5**
- 2.1 Taxonomia lui Flynn . . . 5
- 2.2 Tipuri de Arhitecturi Paralele . . . 6
  - 2.2.1 Shared Memory Multiprocessor (SMP) . . . 6
  - 2.2.2 Distributed Memory Multiprocessor . . . 6
  - 2.2.3 Arhitecturi Hibride . . . 6
- 2.3 Paralelism în Interiorul CPU . . . 6
  - 2.3.1 Instruction-Level Parallelism (ILP) . . . 6
  - 2.3.2 Tipuri de Paralelism . . . 6
- 2.4 Arhitecturi Orientate pe Date . . . 6
  - 2.4.1 Vector Processing . . . 6
  - 2.4.2 SIMD (Single Instruction Multiple Data) . . . 7
  - 2.4.3 Dataflow Architectures . . . 7
- 2.5 Coerența Cache . . . 7
  - 2.5.1 Metode de Implementare . . . 7

3 Rețele de Interconectare **7**
- 3.1 Proprietăți Topologice . . . 7
- 3.2 Strategii de Rutare și Comutare . . . 8
  - 3.2.1 Componente Principale . . . 8
  - 3.2.2 Metode de Rutare . . . 8
- 3.3 Topologii de Rețea . . . 8
- 3.4 Exemple de Sisteme HPC . . . 8
  - 3.4.1 SGI Altix UV . . . 8

3.4.2 Sandia Red Storm (Cray XT3) . . . 8
3.4.3 LLNL BG/L (IBM BlueGene) . . . 8

4 Modele de Programare Paralelă și MPI **9**
4.1 Tipuri de Modele de Calcul Paralel . . . 9
4.2 Ce Este MPI? (Message Passing Interface) . . . 9
4.3 Structura unui Program MPI . . . 9
4.4 Comunicare de Bază în MPI . . . 10
4.4.1 Concepte Fundamentale . . . 10
4.4.2 Funcții Send/Receive . . . 10
4.5 Moduri de Trimitere MPI . . . 10
4.6 Blocaje în MPI . . . 10
4.7 Operații Colective . . . 10
4.8 Când se Utilizează MPI . . . 11

5 Metricile de Performanță în Calculul Paralel **11**
5.1 Măsurători de Timp . . . 11
5.2 Modele de Comunicare . . . 11
5.3 Speedup (Accelerare) . . . 11
5.4 Legile Amdahl și Gustafson . . . 12
5.4.1 Legea lui Amdahl (pesimistă) . . . 12
5.4.2 Legea lui Gustafson (optimistă) . . . 12

1 Concepte Fundamentale de Concurență și Procese

1.1 Terminologie

Concepte de bază

- **Interleaving (Intercalare):** Mai multe sarcini sunt active, dar doar una rulează la un moment dat
- **Multitasking:** Sistemul de operare rulează execuții intercalate
- **Concurrency (Concurență):** Include multiprocesare, multitasking sau orice combinație a acestora

1.2 Procese

Un **proces** este o instanță a unui program în execuție. Sistemele de operare implementează procesele cu o structură tipică ce include:

- **Identificator de proces:** Un ID unic
- **Starea procesului:** Activitatea curentă (nou, rulare, blocat, gata, terminat)
- **Contextul procesului:** Contorul de program, valorile registrelor
- **Memorie:** Textul programului, date globale, stivă și heap

Scheduler-ul este un program de sistem care controlează procesele care rulează, setându-le stările. Un proces poate deveni "blocat" așteptând un eveniment.

1.3 Fire de Execuție (Threads)

Un **thread** este o parte a unui proces al sistemului de operare.

1.3.1 Componente Thread

Private (per thread)	Partajate (între thread-uri)
Identificator de thread	Textul programului
Stare de thread	Date globale
Context de thread	Heap-ul
Stiva (memory: only stack)	

Tabela 1: Componente ale firelor de execuție

1.3.2 Fire de Execuție în C++

Firele de execuție C++ (`std::thread`), introduse în C++11, încapsulează și gestionează un singur fir de execuție.

Important

"All code is accessible to any thread; two threads could be executing the same function at the same time."

Gestionarea în C++:

- **Creare:** `std::thread t(function_name, args...);`
- **join():** Thread-ul apelant (ex: main) așteaptă ca thread-ul specificat să-și termine execuția. Aceasta este "mandatory unless detach is called"
- **detach():** Separă execuția thread-ului de obiectul thread, permițând execuției să continue independent
- **Funcții utile (`std::this_thread`):** `sleep_for()`, `sleep_until()`, `yield()`, `get_id()`

1.4 Partajarea Datelor și Condiții de Cursă

1.4.1 Variabile și Parametri

- **Variabile Globale:** Toate variabilele globale și statice inițializate la compilare sunt accesibile thread-urilor
- **Parametri prin Valoare vs. Referință:** Toți parametrii trecuți unei funcții la pornirea unui thread sunt trecuți prin valoare. Pentru a trece prin referință, este necesară împachetarea explicită în `std::ref()`
- **static vs. thread_local:** Variabilele static sunt inițializate o singură dată și partajate între thread-uri. Pentru a avea o instanță separată a unei variabile statice per thread, trebuie folosită specificația `thread_local`

1.4.2 Condiții de Cursă (Race Conditions)

Definiție Race Condition

"A condition of a program where its behavior depends on relative timing or interleaving of multiple threads or processes."

Tipuri de condiții de cursă:

- **Data Race:** "Occurs when two or more threads access the same variable concurrently, and at least one of the accesses is a write"
- **Condiție de Cursă Critică:** Ordinea operațiilor poate schimba starea finală
- **Condiție de Cursă Non-Critică:** Ordinea operațiilor nu schimbă starea finală

Exemplu: Incrementarea unei variabile partajate (`*x += 1`) de către multiple thread-uri fără sincronizare. Intercalarea operațiilor de citire, adunare și scriere la nivel de registru poate duce la rezultate incorecte.

1.5 Soluții pentru Condițiile de Cursă

1.5.1 Instrucțiuni Atomice

Operații care se execută indivizibil, prevenind interferența altor thread-uri.

Performanță

"Atomic instructions are generally slower and more expensive (in terms of energy) to execute than normal instructions."

Exemplu în C++: `std::atomic<unsigned>* x; x->fetch_add(1);`

1.5.2 Mutexes (Mutual Exclusion)

Principiu Mutex

"At most one thread accesses the shared data at a time!"

- `std::mutex`: Are o stare internă (blocat/deblocat). `lock()` așteaptă până starea devine deblocată, apoi o setează atomic la blocat. `unlock()` setează starea la deblocat
- `std::unique_lock`: O împachetare RAII pentru blocare exclusivă, asigurând deblocarea automată
- **Mutexuri Recursive (`std::recursive_mutex`):** Permit aceluiași thread să blocheze mutex-ul de mai multe ori fără blocare

1.6 Blocaje (Deadlocks)

Apar când două sau mai multe thread-uri se așteaptă reciproc să elibereze resurse.

Soluție pentru Deadlocks

"Deadlocks can be avoided by always locking mutexes in a globally consistent order."

1.7 Variabile Condiție (Condition Variables)

Un primitiv de sincronizare care permite thread-urilor să aștepte până când o condiție arbitrară devine adevărată.

Funcții principale:

- `wait()`: Deși necesită un mutex, îl deblochează și așteaptă variabila condiție
- `notify_one()` / `notify_all()`: Notifică thread-uri în așteptare; mutex-ul nu trebuie ținut de apelant

Spurious Wake-ups

"A notified thread must check the condition explicitly because spurious wake-ups can occur."

2 Arhitecturi Paralele

2.1 Taxonomia lui Flynn

Clasifică arhitecturile de calcul pe baza modului în care procesorul și fluxurile de instrucțiuni/date sunt gestionate:

Tip	Descriere
SISD	Single Instruction, Single Data - Procesor unic
SIMD	Single Instruction, Multiple Data - Procesoare vectoriale, GPU-uri
MISD	Multiple Instruction, Single Data - Rar întâlnit
MIMD	Multiple Instruction, Multiple Data - Majoritatea sistemelor paralele moderne

Tabela 2: Taxonomia lui Flynn

2.2 Tipuri de Arhitecturi Paralele

2.2.1 Shared Memory Multiprocessor (SMP)

- Spațiu de adresare partajat
- Sisteme de memorie bazate pe bus sau rețele de interconectare
- "Cores can be hardware multithreaded (hyperthread)"

2.2.2 Distributed Memory Multiprocessor

- Fiecare procesor are propria memorie privată
- Comunicarea se face prin trecere de mesaje (message passing) între noduri
- Include Massively Parallel Processor (MPP) cu multe procesoare

2.2.3 Arhitecturi Hibride

- **Cluster de SMP-uri:** Memorie partajată în cadrul nodurilor SMP, trecere de mesaje între noduri
- **Multicore SMP + GPU Cluster:** Noduri SMP cu acceleratoare GPU atașate

2.3 Paralelism în Interiorul CPU

2.3.1 Instruction-Level Parallelism (ILP)

- **Pipelining:** Instrucțiunile sunt împărțite în etape (Fetch, Decode, Execute, Memory, Write-back) și executate concurent
- **Superscalar Architecture:** Procesoarele pot executa două sau mai multe instrucțiuni simultan
- **Superpipelining:** Împarte instrucțiunile în mai multe "conducte" separate
- **Very Long Instruction Word (VLIW):** Arhitecturi care permit executarea instrucțiunilor în paralel
- **Hardware Multithreading (Hyperthreading):** Permite unui singur nucleu să execute multiple fire de execuție concurent

Beneficiul Pipelining

"Using instruction pipelining, the instruction throughput increase (instructions per seconds)."

2.3.2 Tipuri de Paralelism

- **Data Parallelism:** Creșterea cantității de date operate simultan
- **Processor Parallelism:** Creșterea numărului de procesoare

2.4 Arhitecturi Orientate pe Date

2.4.1 Vector Processing

Instrucțiunile operează pe valori vectoriale (seturi de date scalare), utilizând registre vectoriale. Exemplu: Cray-1.

2.4.2 SIMD (Single Instruction Multiple Data)

- "Logical single thread (instruction) of control"
- Un controler emite instrucțiuni, iar toate procesoarele le execută
- Exemple: AMT DAP 500, Thinking Machines Connection Machine

2.4.3 Dataflow Architectures

- Reprezintă calculul ca un graf de dependențe
- Operațiile sunt executate când operanzii sunt pregătiți ("tokens")
- "Machine does the hard parallelization work"
- Dar sunt "hard to build correctly !!!!"

2.5 Coerența Cache

Asigură că multiple copii ale aceluiași bloc de memorie (în cache-uri diferite) rămân consistente.

2.5.1 Metode de Implementare

Snooping (Spionaj):

- Controlerul cache monitorizează toate tranzacțiile pe bus-ul partajat
- Protocoale: MESI (Modified, Exclusive, Shared, Invalid) este un protocol comun

Directoare:

- Abordare scalabilă unde fiecare bloc de memorie are informații de director asociate
- Comunicarea se face prin tranzacții de rețea
- Exemplu: Stanford DASH Multiprocessor

3 Rețele de Interconectare

3.1 Proprietăți Topologice

Metrici importante

- **Distanța de rutare:** Numărul de legături pe ruta de la sursă la destinație
- **Diametrul:** Distanța maximă de rutare
- **Lățimea de bi-secțiune:** Numărul minim de legături ce trebuie tăiate pentru a împărți rețeaua în două grafuri disconexe de dimensiuni egale
- **Scalabilitate:** Abilitatea de a fi extinsă pentru a găzdui cantități crescânde de muncă

Bisection Bandwidth

"Bisection bandwidth accounts for the bottleneck bandwidth of the entire network."

3.2 Strategii de Rutare și Comutare

3.2.1 Componente Principale

- **Routing Algorithm:** Restricționează setul de căi pe care le pot urma mesajele
- **Switching Strategy:** Modul în care datele traversează o rută
- **Flow Control Mechanism:** Când un mesaj traversează o rută și ce se întâmplă la întâlnirea traficului

3.2.2 Metode de Rutare

- **Store-and-Forward Routing:** Un mesaj este complet primit la un hop intermediar înainte de a fi retransmis
- **Cut-Through Routing:** Mesajele sunt împărțite în "flits" mici. Reduce latența

3.3 Topologii de Rețea

Topologie	Diametru	Caracteristici
Completely Connected	1	Cost O(p^2)
Linear Array	p-1	Simplu, dar ineficient
2-D Mesh	$2(\sqrt{p}-1)$	Fără wraparound
2-D Mesh (wraparound)	$2(\sqrt{p}/2)$	Cu wraparound
Hypercube	log p	log p vecini per nod
Trees	log p	Fără încrucișări în 2D
Fat Trees	log p	Bandwidth crescătoare

Tabela 3: Comparație topologii de rețea

3.4 Exemple de Sisteme HPC

3.4.1 SGI Altix UV

- Arhitectură de memorie partajată scalabilă
- Până la 2048 de nuclee (arhitectural până la 262,144)
- 16TB de memorie partajată globală
- Interconnect NUMAlink 5

3.4.2 Sandia Red Storm (Cray XT3)

- Arhitectură distribuită cu 12,960 procesoare AMD Opteron
- Interconnect mesh 3D
- Utilizează MPI

3.4.3 LLNL BG/L (IBM BlueGene)

- Sistem masiv cu 65,536 noduri dual-procesor
- Rețea torus 3D (32x32x64)
- MPI

4 Modele de Programare Paralelă și MPI

4.1 Tipuri de Modele de Calcul Paralel

- **Data Parallel:** Aceleași instrucțiuni sunt executate simultan pe multiple elemente de date (SIMD)
- **Task Parallel:** Instrucțiuni diferite pe date diferite (MIMD)

4.2 Ce Este MPI? (Message Passing Interface)

Definiția MPI

"A message-passing library specification" – nu este un limbaj sau un compilator.

Caracteristici principale:

- Proiectat pentru calculatoare paralele, clustere și rețele eterogene
- Oferă o modalitate "powerful, efficient, and portable" de a exprima programe paralele
- Pentru comunicarea între procese cu "separate address spaces"
- Compatibil pentru Distributed Memory (DM), Shared Memory (SM) și Hybrid
- Are binding-uri pentru C, C++ și Fortran

4.3 Structura unui Program MPI

Listing 1: Structura de bază MPI

```
#include <mpi.h>

int main(int argc, char** argv) {
    int numprocs, myid;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    // Codul programului

    MPI_Finalize();
    return 0;
}
```

Funcții principale:

- `MPI_Init(&argc, &argv)`: Inițializează mediul MPI
- `MPI_Comm_size(MPI_COMM_WORLD, &numprocs)`: Obține numărul total de procese
- `MPI_Comm_rank(MPI_COMM_WORLD, &myid)`: Obține rank-ul procesului curent
- `MPI_Finalize()`: Oprește mediul MPI

Compilare și execuție:

- Compilare: `mpicc hello.c -o hello`
- Execuție: `mpirun -np 4 hello`

4.4 Comunicare de Bază în MPI

4.4.1 Concepte Fundamentale

- **Comunicator (MPI_COMM_WORLD):** Conține toate procesele inițiale
- **Datatypes (MPI_Datatype):** Descriu datele dintr-un mesaj (MPI_INT, MPI_DOUBLE)
- **Tags:** Numere întregi pentru identificarea mesajelor

4.4.2 Funcții Send/Receive

- `MPI_Send(start, count, datatype, dest, tag, comm)`: Funcție de trimitere blocantă
- `MPI_Recv(start, count, datatype, source, tag, comm, status)`: Funcție de primire blocantă
- `MPI_Get_count(&status, datatype, &recvd_count)`: Extrage numărul de elemente primite

4.5 Moduri de Trimitere MPI

Funcție	Comportament
MPI_Send	Poate bloca sau nu
MPI_Bsend	Returnează imediat (Buffered)
MPI_Ssend	Nu returnează până la primire (Synchronous)
MPI_Rsend	Doar dacă primirea e postată (Ready)
MPI_Isend	Non-blocking, necesită MPI_Wait

Tabela 4: Moduri de trimitere MPI

4.6 Blocaje în MPI

Deadlock clasic

Procesele 0 și 1 trimit reciproc mesaje mari simultan (Send(1) urmat de Recv(1)). Dacă nu există suficient spațiu de buffering, pot apărea blocaje.

Soluții:

- **Order the operations:** Procesul 0 trimite, Procesul 1 primește, apoi invers
- **Non-blocking operations:** MPI_Isend și MPI_Irecv urmate de MPI_Waitall
- **MPI_Sendrecv:** Operație atomică care evită blocajele

4.7 Operații Colective

- **MPI_Bcast:** Procesul root trimite date tuturor celorlalte procese
- **MPI_Reduce:** Combină date de la toate procesele într-un singur rezultat
- **MPI_Barrier:** Sincronizează toate procesele dintr-un comunicator

Sincronizare

"A user program that assumes no synchronization is erroneous."

4.8 Când se Utilizează MPI

Adecvat pentru:

- Portabilitate și Performanță
- Construirea de Instrumente (biblioteci)
- Necesitatea de a gestiona memoria pe bază de procesor

Nu este adecvat pentru:

- Necesitatea toleranței la erori (fault tolerance)
- Computing Distribuit (CORBA, DCOM etc.)

5 Metricile de Performanță în Calculul Paralel

5.1 Măsurători de Timp

- **Wall clock time:** Timpul de la începutul primului procesor până la oprirea ultimului procesor
- **FLOPS:** Floating Point Operations Per Second - măsură brută a performanței

5.2 Modele de Comunicare

Timpul de comunicare

$$T_{comm} = t_s + N_w \cdot t_w$$

unde:

- t_s (startup time): Timpul petrecut la nodurile de trimitere și primire
- N_w (number of words): Numărul de cuvinte dintr-un mesaj
- t_w (per-word transfer time): Timpul de transfer per cuvânt
- t_h (per-hop time): Timp în funcție de numărul de hop-uri

5.3 Speedup (Accelerare)

Formula Speedup

$$Speedup = \frac{T_S}{T_P}$$

unde T_S este timpul de execuție al celui mai bun algoritm secvențial, iar T_P este timpul de execuție al programului paralel.

Tipuri de Speedup:

- **Relativ Speedup:** T_S este timpul programului paralel executat pe un singur procesor
- **Absolute Speedup:** T_S consideră cel mai bun algoritm secvențial

Limita Superioară

Speedup-ul, în teorie, nu poate depăși numărul de procesoare (n). Un speedup mai mare decât n este posibil doar prin "time sliding", ceea ce ar contrazice ipoteza că T_S este timpul celui mai rapid program secvențial.

5.4 Legile Amdahl și Gustafson

5.4.1 Legea lui Amdahl (pesimistă)

Formula Amdahl

$$Speedup = \frac{1}{seq + \frac{par}{n}}$$

unde *seq* este fracția secvențială, *par* este fracția paralelă, și n este numărul de procesoare.

"When n $\rightarrow$ infinity, then the speedup approaches to 1/seq."

Afirmă că speedup-ul este limitat de partea secvențială a programului, presupunând o dimensiune fixă a problemei.

5.4.2 Legea lui Gustafson (optimistă)

Formula Gustafson

$$Speedup = seq(m) + n \cdot (1 - seq(m))$$

unde m este dimensiunea problemei.

Ia o "opposite view" față de Amdahl, susținând că, pe măsură ce dimensiunea problemei crește, partea secvențială va deveni un procent din ce în ce mai mic din întregul proces.