Exercitii

Monday, January 27, 2025 1:43 PM

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).

Pentru calcul de tip arbore binar
1024 = 2^10 => un arbore binar cu 10 niveluri
Tp = log2(1024)(tc + tcom) = 10 * (1 + 0) = 10

Ts = 1024

Acceleratie = Ts/Tp = 1024/10 = 102.4

Eficienta = Sp/p = 102.4/1024 = 0.1

Cost = Tp*p = 10*1024 = 10240

[10, 102.4, 0.1, 10240]

Exemplu: Adunare n numere pe un model distribuit

- $T_P = (t_c+t_{com})\ \log n$ (pt $p = n$).

t_c timpul necesar unei operatii de adunare;
t_{com-} timpul necesar unei operatii de comunicatie;

- $C = p\ T_P = O(n \log n)$
- $T_s = \Theta(n)$ => nu este cost optimal
- Cum se poate optimiza?
  - se micsoreaza p; **p=n/k**
  - se considera **p** segmente (**dim = n/p**) pentru care se calculeaza suma secvential
  - se foloseste calculul de tip arbore pentru insumarea celor p sume locale
  - $T_p = t_c\ n/p + (t_c+t_{com}) \log p$
  - $C = t_c\ n+(t_c+t_{com})p*\log p$ => daca $p*\log p = O(n)$ atunci cost optimal

Schita implementare Semafor

class Semaphore
count: INTEGER
down
do

```
                await count > 0
                count := count - 1
        end
    up
        do
                count := count + 1
        end
end
```

Sau pentru procese

```
class Semaphore
    count: INTEGER
    blocked: CONTAINER / QUEUE
    down
        do
            if count > 0 then
                count := count - 1
            else
                blocked.add(P)
                P.state = blocked
            end
        end
    up
        do
            if blocked.is_empty then
                count := count + 1
            else
                Q = blocked.remove // select some process Q
                Q.state = ready
            end
        end
```

Implementare Java: clasa Semaphore din java.util.concurrent.Semaphore
Implementare C++: Ig pentru semafor binar mutex din <mutex>

Schita implementare Monitor

```
class Monitor
    feature
    -- attribute declarations
    a1: Type1

    -- routine declarations
    r1(arg1, …, argk) do … end

    invariant
    -- monitor invariant
```

Ceva ceva folosind un semafor binar: entry.

r(arg1, …, argk)

```
        do
                entry.down
                body
                entry.up
        end
```

Monitor in Java

```
// metodele monitorului
synchronized type m(args) {
        … // body
}
```

Schita implementare Variabila Conditionala (cu Monitor?)

```
class Condition_Variable
        feature
        blocked: Queue
        entry: Monitor // ?
        wait
                do
                        entry.up // entry obiect de tip monitor ig
                        blocked.add(P)
                        P.state := blocked
                end
        signal deferred end

        is_empty: BOOLEAN
                do
                        result := blocked.is_empty
                end
end
```

(Banuiesc ca trebuie inclusa si o metoda de signal)
Signal & Continue

```
signal
        do
                if not blocked.is_empty then
                        Q := blocked.remove
                        entry.blocked.add(Q) // entry este un obiect de tip monitor ig
                end
        end
```

Signal & Wait

```
signal
        do
                if not blocked.is_empty then
                        entry.blocked.add(P)
                        Q := blocked.remove
                        Q.state := ready
```

```
                P.state := blocked
            end
        end
```

CV implementare orientativa
(Lock implementat ca si un semafor binar initializat cu 1)

```
class CV {
        Semaphore s, x;
        Lock m;
         int waiters = 0;
public CV(Lock m) {
// Constructor
        this.m = m;
        s = new Semaphore();
        s.count = 0;        s.limit = 1;
        x = new Semaphore();
        x.count = 1;        x.limit = 1;
        }
// x protejeaza accesul la variabila 'waiters'
```

```
public void Wait() {
// Pre-condition: this thread holds "m"
//=> Wait se poate apela doar dintr-un cod
//sincronizat (blocat ) cu "m"
        x.P(); {
            waiters++; }
        x.V();
        m.Release();
(1)
        s.P();
         m.Acquire();
}
 public void Signal() {
        x.P(); {
         if (waiters > 0)
        {       waiters--;     s.V();          }
        x.V();
         }
}
```

Implementare Java: interfata Condition din java.util.concurrent.locks
Implementare C++: condition_variable din <condition_variable>

ReadWriteLock din java.util.concurrent.locks, din C++ scop similar shared_mutex din <shared_mutex>

CyclicBarrier din java.util.concurrent.CyclicBarrier, din C++ similar barrier din <barrier>

Future-Promise
Implementare Java: **Future**, **CompletableFuture**, Runnable, Callable, Executor, FutureTask, runAsnyc, supplyAsync
Implementare C++: **promise**, **future, shared_future**, async, packaged_task din <future>

SUBIECTE
Exemplu Subiect

1. Definiti conceptul de semafor. Exemplificati.

Un semafor este o primitiva de sincronizare de nivel inalt, poate fi definit ca o pereche {v(s), c(s)}. v(s) - variabila, c(s) - colectie de asteptare. Acesta are 2 operatii:
P(s)/down, este apelata cand un proces doreste sa primeasca acces la o sectiune critica, asteapta pana cand v(s) > 0, si V(s)/up, semnifica eliberarea sectiunii critice pentru alte procese.

2. Cum se defineste accelerarea unei aplicatii paralele? Ce evidentiaza aceasta?

Accelerarea unei aplicatii paralele este Sp(n) = Ts(n)/Tp(n), raportul dintre timpul de executie al celui mai bun algoritm secvential, rulat de un calculator monoprocesor si timpul de executie al programului paralel echivalent.

3. Care este granularitatea aplicatiilor de tip "embarrassingly parallel programs"?

Granularitatea acestor aplicatii este mare/coarse-grain, deoarece raportul dintre timpul de calcul si timpul de comunicare este foarte mare.

4. Schita program multithreading care poate produce deadlock

Thread1
…
lock(a);
// alte operatii
lock(b);

Thread2
…
lock(b);
// alte operatii
lock(a);

5. Analizati secventa de cod. Ce se poate intampla?

b ig

2024_1

1. Ce inseamna situatie de tip "data-race". Cum definiti o sectiune critica? Exemplificati printr-un exemplu concret.

O situatie de tip data-race are loc atunci cand doua sau mai multe threaduri(/procese) incearca sa acceseze aceeasi resursa concurent, iar cel putin unul dintre accesuri este de scriere.
O sectiune critica este o parte din cod unde poate avea loc un data-race.

Exemplu
Amandoua threadurile ruleaza codul

…
a = a + 1 // a - o variabila globala sau primita prin referinta
…

Daca nu se folosesc mecanisme de sincronizare aceasta este o sectiune critica

2. Definiti conceptul de semafor si furnizati o schita de implementare. Evidentiati o implementare existenta in bibliotecile C++ sau Java. Exemplificati modul de folosire printr-un exemplu concret (cod).

Concept -||-
O implementare existenta in Java: clasa Semaphore din java.utils.concurrent.Semaphore.
Exemplu

```
class MyThread extends Thread {
Sempahore sem;

MyThread(Semaphore sem) {
	this.sem = sem;
}

public void run() {
	sem.acquire();
	// other operations
	sem.release();
}

//
int limit = 1;
Semaphore sem = new Semaphore(limit);
MyThread mt1 = new MyThread(sem);
MyThread mt2 = new MyThread(sem);

mt1.run();
mt2.run();

mt1.join();
mt2.join();
```

3. Sablonul de programare paralela Pipeline: prezentare generale, avantaje, operatiile care se paralelizeaza. Dati un exemplu de program care respecta acest sablon, evidentiati principalele etape ale programului si faceti analiza teoretica a complexitatii timp, a acceleratiei si a gradului de paralelizare pentru exemplul considerat.

Sablonul de programare paralela Pipeline reprezinta o secventa de stagii care transforma un flux de date. Un proces este impartit in mai multe stagii, care pot fi executate individual. Paralelizarea pipeline se face prin executia diferitelor stagii in paralel si executia multiplelor copii ale stagiilor fara stare in paralel.
Avantaje: performanta crescuta, reducerea latentei.
Operatiile care se paralelizeaza: transformarea datelor.

Un pogram care respecta acest sablon este un program care proceseaza un flux de numere si vrea sa le transforme aplicand diferite operatii asupra acestora, de exemplu pentru x, sa obtina rezultatul x * 2 + 15.
Acest proces poate fi impartit intr-un task care inmulteste fluxul de date cu 2, si al task care aduna 15 la fluxul de la taskul anterior.

Complexitatea timp este pentru n date in jur de n/2 deci O(n)

Din moment ce cele 2 taskuri se pot executa in paralel, timpul programului paralel este de 2 ori mai rapid decat cel secvential.
Sp = Ts/Tp = 2

Gradul de paralelizare este 2, din moment ce se pot efectua 2 operatii in acelasi timp.

4. Prin ce se caracterizeaza scalabilitatea unei aplicatii paralele? Cand spunem despre o aplicatie ca este scalabila? Dati un exemplu(program) de aplicatie scalabila.

Scalabilitatea unei aplicatii reprezinta abilitatea unei aplicatii de a obtine o crestere de performanta proportionala cu numarul de procesoare.
Spunem despre o aplicatie ca este scalabila atat timp cat respecta aceasta caracteristica, iar cresterea numarului de procesoare (pana la un numar maxim de care se poate utiliza aplicatia) creste performanta.

Un exemplu de aplicatie scalabila: Idk adunare de vectori.

5. Evidentiati caracteristicile generale ale unei aplicatii "peer-to-peer". Prin ce se deosebeste o aplicatie "peer-to-peer" de o aplicatie "client-server"

O aplicatie "peer-to-peer" este formata dintr-o retea de noduri interconectate intre ele. Un nod(peer) poate functiona ca un client - cerand servicii de la alte componente, sau ca un server - furnizeaza servicii pentru altii.
Performanta creste atunci cand exista un numar mare de noduri, dar scade atunci cand sunt prea putine. O aplicatie de tip "client-server" are un singur server si mai multi clienti pot cere informatii de la server. Aplicatiile p2p sunt mai scalabile, si au un cost de administrare mai scazut fata de aplicatiile c-s, dar nu pot garanta calitatea serviciilor sau calitatea securitatii.

6. Program MPI…

2024_2

1. Sincronizare prin bariera de sincronizare. Ce inseamna o bariera de sincronizare? Schita de implementare.

O bariera de sincronizare este un mecanism care opreste executia tututor threadurilor care ajung la aceasta pana cand se aduna la fel de multe threaduri cu un numar dat barierei.

```
class Barrier {
	threadCount: INTEGER
	waiting: QUEUE

	Barrier(threadCount: INTEGER) {
		this.threadCount = threadCount
	}

	await(T: Thread) {
		waitingThreads++

		if (waiting.size() < threadCount) {
			waiting.push(T)
			T.state = waiting
		} else {
			while (!waiting.is_empty()) {
				T = waiting.pop()
				T.state = ready
				}
		}
	}
}
```

2. Definiti conceptul de monitor, necesitatea si avantajele oferite de acesta. Explicati cum

este folosit in Java acest concept. Exemplificati modul de folosire a conceptului de monitor in Java printr-un exemplu concret

Un monitor poate fin considerat un tip abstract de data care consta din: un set permanent de variabile ce reprezinta resursa critica, un set de proceduri ce reprezinta operatii asupra variabilelor si un corp de initializare. Dupa initializare monitorul poate fi accesat doar prin metodele lui. Doar una dintre procedurile monitorului poate fi executata la un moment dat, din aceasta cauza un avantaj al monitorului este rezolvarea problemelor de tip data-race sau sectiune critica.

In Java fiecare obiect are un monitor.

```
class Monitor {
	int a = 0; // resursa critica

	public void synchronized increment() {
		a++;
	}

	public void synchronized decrement() {
		a--;
	}
}
```

3. Sablonul de programare paralele Divide&Impera - prezentare generala, avantaje, operatiile care se pot paraleliza. Exemplu de program care respecta acest sablon, faceti analiza teoretica a complexitatii timp, acceleratiei si a gradului de paralelizare.

Acest sablon este folosit, in general, pentru probleme care se pot rezolva cu Divide&Impera, astfel problema se imparte in diferite subprobleme care se poate prelucra separat, iar rezultatele sunt combinate pentru a obtine un rezultat final.

4. Aplicatie cu granularitate unitara (1)????

Adunare de vector cu dimensiunea n pe n threaduri diferite, unde fiecare thread face o adunare ???

5. Acceleratia unei aplicatii paralele. Ce evidentiaza? Enuntati si explicati Legea lui Amdhal si Legea lui Gustafon.

Acceleratia este definita ca raportul dintre timpul de executie al celui mai rapid algoritm serial cunoscut, executat pe un calculator monoprocesor, si timpul de executie al programului paralel echivalent, executat pe un sistem paralel.
$Sp(n) = Ts(n)/Tp(n)$

Legea lui Amdhal
Accelerarea procesarii depinde de raportul partii secventiale fata de cea paralelizabila
$Speedup = 1/(seq + par/p)$

Legea lui Gustafon
Atunci cand dimensiunea problemei creste, partea seriala se micsoreaza in procent
$Speedup = seq(m) + p*par(m)$