1.Ce este un monitor? Dati exemplu in Java.

Un monitor poate fi considerat un tip abstract de dată (poate fi implementat ca si o clasa) care constă din:

– un set permanent de variabile ce reprezintă resursa critică,

– un set de proceduri ce reprezintă operaţii asupra variabilelor şi

– un corp (secvenţă de instrucţiuni).

Exemplu:

Object lock = new Object();

synchronized (lock) {

// critical section

}

:

synchronized type m(args) {

// body

}

• echivalent

type m(args) {

synchronized (this) {

// body

}

}

• Corpul este apelat la lansarea ‘programului’ şi produce valori iniţiale pentru variabilele-monitor (cod de initializare).

• Apoi monitorul este accesat numai prin procedurile sale.

2.Faceti schita unu program mpi ce rezolva adunarea a doua matrici de nxn.Calculati costul, complezitatea timp, acceleratia si eficienta.Este solutia aleasa optima d.p.d.v. al costului? PAS PAS

3. Ce este granularitatea unui program? Cum este granularitatea aplicatiei"embarrassingly parallel programs" ?(paralelizarea triviala)

Granularitatea(“grain size”) este un parametru calitativ care caracterizeaza atat – sistemele paralele cat si – aplicatiile paralele.

Granularitatea aplicatiei se defineste ca dimensiunea minima a unei unitati secventiale dintr-un program, exprimata ın numar de instructiuni. – Prin unitate secventiala se ıntelege o parte programın care nu au loc operatii de sincronizare sau comunicare cu alte procese.

Granularitatea se referă la mărimea task-ului în comparație cu timpul necesar comunicației și sincronizării datelor. Granularitatea paralelizarii triviale poate fii coarse-grained sau fine-grained.

Fine-grain Parallelism: – Relatively small amounts of computational work are done between communication events – Low computation to communication ratio – Facilitates load balancing

– Implies high communication overhead and less opportunity for performance enhancement

Coarse-grain Parallelism: – Relatively large amounts of computational work are done between communication/synchronization events – High computation to communication ratio – Implies more opportunity for performance increase – Harder to load balance efficiently

Subiectul 4 de teorie:

1. Var cond

– O abstractizare care permite sincronizarea conditionala;

Operatii: wait; signal ; [broadcast]

– O variabila conditionala C este asociata cu o variabila de tip Lock – m

• Thread t apel wait =>

– suspenda t si il adauga in coada lui C + deblocheaza m (op atomica)

• Atunci cand t isi reia executia m se blocheaza

• Thread v apel signal =>

– se verifica daca este vreun thread care asteapta si il activeaza

Legatura cu monitor: – Variabile conditionale pot fi asociate cu lacatul unui monitor (monitor lock);

• Permit threadurilor sa astepte in interiorul unei sectiuni critice eliberand lacatul monitorului.

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’

2. Costul + o schita parca pentru un program cu n numere si procese nu mai stiu sigur.

Costul se defineste ca fiind produsul dintre timpul de executie ¸si numarul maxim de procesoare care se folosesc: Cp(n) = tp(n) · p.

• O aplica=e paralela este op4ma din punct de vedere al costului, daca valoarea acestuia este egala, sau este de acelasi ordin de marime cu =mpul celei mai bune variante secven=ale; • aplica=a este eficienta din punct de vedere al costului daca Cp = O(t1 log p).

Costul unui sistem paralel (algoritm +sistem) • Cost = p x TP

SCHITA ^ - asa am gasit , nu stiu sigur daca este ok.

Adunare n numere cu p procesoare (ambele sunt puteri ale lui 2) • Fiecarui procesor dintre cele p ii sunt atribuite n / p procesoare virtuale. • Primii log n - log p din cei log n pasi ai algoritmului original se simuleaza folosind p procesoare in Θ ( (n / p) (log n - log p) )= Θ ((n / p) log (n/p)) • Urmatorii log p pasi nu necesita nici par77onare (p noduri – p procesoare) • T\_p = Θ ( (n / p) log (n/p) +log p ) • C = O (n log n), • T\_s = Θ (n) => Sistemul paralel nu este cost op=mal

3.race condition si zona critica

Race condition are loc atunci cand la executie exista interactiune intre threaduri/procese si rezultatul depinde de interleaving, pot fi extrem de greu de depistat.

• O sectiune de cod care conduce la race conditions se numeste critical section (sectiune critica).

public class Counter {

protected long count = 0;

public void add(long value){

this.count = this.count + value;

}

}

Soluţii: o atomicizarea zonei critice o dezactivarea preempţiei în zona critică o secvenţializarea accesului la zona critică

Sub 2 teorie:

1. sablonul divide et impera, exemple, gradul de paralelism

Divide&impera este bine cunoscuta din dezvoltarea algoritmilor secventiali. O problema este ımpartita ın doua sau mai multe subprobleme. Fiecare dintre aceste subprobleme este rezolvata independent si rezultatele lor sunt combinate pentru a se obtine rezultatul final.

Exemple: sortare prin interclasare:

Procedure interSort(A, n)

if (n > 1) then

imparte(A, n, A0, n0, A1, n1);

in parallel

interSort(A0, n0),

interSort(A1, n1)

end in parallel

combina(A0, n0, A1, n1, A, n);

end if

cautare paralela

Function CautaParalel(A, n, x)

if (n > m) then

imparte(A, n, A[0], n[0], A[1], n[1], A[2], n[2]);

for i = 0, 2 in parallel do

c[i] ← CautaParalel(A[i], n[i], x);

end for

if (c[0] != −1) then

CautaParalel← c[0];

else

if (c[1] != −1) then

CautaParalel← n[0] + c[1];

else

if (c[2] != −1) then

CautaParalel← n[0] + n[1] + c[2];

else

CautaParalel← −1;

end if

end if

end if

else

CautaParalel ← CautaSecvential(A, n, x);

end if

2. dedalocks pe threads, procese

• Deadlock – situatia in care un grup de procese/threaduri se blocheaza la infinit pentru ca fiecarea proces asteapta dupa o resursa care este retinuta de alt proces care la randul lui asteapta dupa alta resursa.

public class TreeNode {

TreeNode parent = null;

List children = new ArrayList();

public synchronized void addChild(TreeNode child)

{ if(! this.children.contains(child))

{ this.children.add(child);

child.setParentOnly(this);

} }

public synchronized void addChildOnly(TreeNode child)

{ if(!this.children.contains(child)

{ this.children.add(child); } }

public synchronized void setParent(TreeNode parent){ this.parent = parent; parent.addChildOnly(this); } public synchronized void setParentOnly(TreeNode parent){ this.parent = parent; } }

3. wait notify notifyAll in java

Wait() suspenda threadul si deblocheaza operatia atomica

notify() deblocheaza un proces arbitrar.

Java "monitors" nu sunt starvation-free – notify() deblocheaza un proces arbitrar.

notifyAll() trezesc toate firele care așteaptă pe monitorul acestui obiect

• Apelurile metodelor notify() si notifiAll() nu se salveaza in cazul in care nici un thread nu asteapta atunci cand sunt apelate.

Asfel semnalul notify()se pierde.

Acest lucru poate conduce la situatii in care un thread asteapta nedefinit, pentru ca mesajul corespunzator de notificare se pierde.

Sub 3:

1. Semafoare

Semaforul este primitive de sincronizara de nivel inalt. Inventata de E.W. Dijkstra in 1965 .Este caracterizat de o variabina count=v(s) (val semafor) si 2 operatii V(s)/up si P(s)/down.

– P(s) –este apelată de către procese care doresc să acceseze o regiune critică pt a obţine acces.

• Efect: - incercarea obtinerii accesului procesului apelant la secţiunea critică si decrementarea valorii.

- dacă v(s) <= 0 , procesul ce doreşte execuţia sectiunii critice aşteaptă

– V(s)

• Efect : incrementarea valorii semaforului.

• se apelează la sfârşitul secţiunii critice şi semnifică eliberarea acesteia pt. alte procese.

Semafor Binar • Valoarea semaforului poate lua doar valorile 0 si 1 Valoarea =>poate fi de tip Boolean

Daca semaforul se foloseste fara a se mentine o evidenta a proceselor care asteapta intrarea in sectiunea critica nu se poate asigura starvation-free

Un semafor ‘slab’ se poate defini ca o pereche {v(s),c(s)} unde: -v(s) este valorea semaforului- un nr. întreg a cărui valoare poate varia pe durata execuţiei diferitelor procese. -c(s) o multime de asteptare la semafor - conţine referinţe la procesele care aşteaptă la semaforul s. + Operatiile P(s)/down si V(s)/up

Un semafor ‘puternic’ se poate defini ca o pereche {v(s),c(s)} unde: -v(s) este valorea semaforului- un nr. întreg a cărui valoare poate varia pe durata execuţiei diferitelor procese. -c(s) o coadă de aşteptare la semafor - conţine referinţe la procesele care aşteaptă la semaforul s (FIFO). + Operatiile P/down si V/up

2. Eficienta si alea alea la o problema din aia trivial

Aproximarea numarului π Un calcul de tip Monte Carlo se realizeaza pentru aproximarea numarului π prin urmatoarea metoda: Se considera un cerc de raza egala cu unitatea ınscris ıntr-un patrat.

Complexitatea acestui algoritm este O(n), unde n este numarul de puncte generate aleator de fiecare componenta. Varianta secventiala a acestui calcul are complexitatea O(np) ¸si prin urmare acest algoritm este un algoritm foarte eficient: Ep(n) ≈ 1.

3. Scalabilitate

Principala proprietate a sistemelor cu memorie distribuita, care le avantajeaza fata de cele cu memorie comuna, este scalabilitatea Scalabilitate aplica+e: abilitatea unui program paralel sa ob+na o crestere de performanta propor+onal cu numarul de procesoare si dimensiunea problemei.

Scalabilitatea masoara modul ın care se schimba performanta unui anumit algoritm ın cazul ın care sunt folosite mai multe elemente de procesare.

• Scalabilitatea unui sistem paralel este o masura a capacitatii de a livra o accelerare cu o crestere liniara in functie de numarul de procesoare folosite.

Sub6:

1. Distributie date si distributie functionala

Exista doua strategii principale de partitionare: – descompunerea domeniului de date si – descompunerea functionala.

In functie de acestea putem considera aplicatii paralele bazate pe: – descompunerea domeniului de date – paralelism de date, si – aplicatii paralele bazate pe descompunerea functionala.

Descompunerea domeniului de date • Este aplicabila atunci cand domeniul datelor este mare si regulat. Ideea centrala este de a divide domeniul de date, reprezentat de principalele structuri de date, ın componente care pot fi manipulate independent. • Apoi se partitioneaza operatiile, de regula prin asocierea calculelor cu datele asupra carora se efectueaza. • Astfel, se obtine un numar de activitati de calcul, definite de un numar de date si de operatii.

Descopunerea functionala este o tehnica de partitionare folosita atunci cand aspectul dominant al problemei este functia, sau algoritmul, mai degraba decat operatiile asupra datelor. • Obiectivul este descompunerea calculelor ın activitati de calcul cat mai fine. • Dupa crearea acestora se examineaza cerintele asupra datelor. • Focalizarea asupra calculelor poate revela uneori o anumita structura a problemei, de unde oportunitati de optimizare, care nu sunt evidente numai din studiul datelor. • In plus, ea are un rol important ca si tehnica de structurare a programelor.

Exista mai multe tehnici de partitionare a datelor, care pot fi exprimate si formal prin functii definite pe multimea indicilor datelor de intrare cu valori in multimea indicilor de procese. • Cele mai folosite tehnici de partitonare sunt prin “taiere” si prin “incretire” care corespund distributiilor liniara si ciclica. Distributia liniara in curs este si cea –distributie bloc

2.acceleratie + legea lui Ambhdal

Acceleratia(“speed-up”), notata cu Sp, este definita ca raportul dintre timpul de executie al celui mai bun algoritm serial cunoscut, executat pe un calculator monoprocesor si timpul de executie al programului paralel echivalent, executat pe un sistem de calccul parallel. Daca se noteaza cu ts timpul de executie al programului serial, iar tp timpul de executie corespunzator programului paralel, atunci: Sp(n) = ts(n) tp(n) . (1.1) Numarul n reprezinta dimensiunea datelor de intrare, iar p numarul de procesoare folosite.

(Legea lui Amdahl) Fie α(0 ≤ α ≤ 1) proportia operatiilor din algoritm care se executa secvential (fractia lui Amdahl). Atunci:

- Partea seriala a algoritmului se executa ın timpul αts.

- Partea paralela a algoritmului se executa ın timpul (1−α)ts p .

- Intregul algoritm se executa ın timpul tp = αts + (1−α)ts p .

- Acceleratia relativa este RSp = (α+ 1−α p ) −1 care nu poate depasi α −1 (Legea lui Amdhal)

3.client server vs peer to peer

Peer-to-peer se bazeaza pe sharingul de date si fisiere pe o retea de useri interconectati, reteaua e mica deci nu exista server in schimb fiecare user actioneaza ca si client si server In acelasi timp. Dezavantajul este securitatea deoarece oricine poate intra daca are parola si ca datele sunt mai instabile depinzand de fiecare membru al retelei in parte. Avantajul este uzul minim de resurse

Client/server se bazeaza cu un server unde sunt stocate fisierele si parolele iar accesul la date este regulat de catre un administrator de retea, de aceea sunt mai sigure, dezavantajul este ca ele pot fi foarte scumpe ca resurse.

s5:

1.Bariere de sincronizare exemplu in mpi

O bariera de sincronizare este un mecanism de baza ın sincronizarea globala. Este introdusa ın punctul ın care fiecare proces trebuie sa le astepte pe celelalte, iar executia se reia doar dupa ce toate procesele au atins bariera.

Exemplu:

MPI\_Barrier

MPI\_Barrier (comm)

MPI\_BARRIER (comm,ierr)

2.sistemele flynn si ce tip de sistem crederi ca e CUDA?explicati .

Clasificarea Flynn Michael J. Flynn în 1966 • SISD: sistem cu un singur flux de instrucţiuni şi un singur flux de date; • SIMD: sistem cu un singur flux de instrucţiuni şi mai multe fluxuri de date; • MISD: sistem cu mai multe fluxuri de instrucţiuni şi un singur flux de date; • MIMD: cu mai multe fluxuri de instrucţiuni şi mai multe fluxuri de date.

Ce este CUDA? • Compute Unified Device Architecture” • o pla7orma de programare paralela-> • Arhitectura care foloseste GPU pt calcul general – permite cresterea performantei • Released by NVIDIA in 2007 • Model de programare – Bazat pe extensii C / C++ - pt a permite ‘heterogeneous programming’ – API pt gesQonarea device-urilor, a memoriei etc.

CUDA foloseste SIMD pt ca poate sa scrie si sa citeasca din memorie direct , pe cand in GPU ai nevoie sa o uploadez inainte de a fi accesata.   
În al doilea rând, atât SIMD, cât și GPU-urile sunt rău la codul extrem de fragil, însă SIMD e mai putin rău. Acest lucru se datorează faptului că GPU-urile grupă mai multe fire (un "warp") sub un singur dispecer de instrucțiuni

3.ce este granularitatea? exemplu aplicatie cu granularitatea ideala=1 & exemplu in mpi

Nu stiu la exemplu aplicatie cu granularitate ideala si mpi.