IX. Designul algoritmilor paraleli (1)

Continut

- Concurența în programele paralele,
- Abordări pentru obținerea concurentei,
- Bazele concurentei în software;
- Sarcini, procese și procesoare,
- Etapele de proiectare,
- Descompunere
  - exemple simple,
  - clasificare.

Exploatarea concurenței

- Cheia pentru calculul paralel este exploatarea concurenței.
- Concurența există într-o problemă de calcul atunci când problema poate fi descompusă în subprobleme care pot fi executate în siguranță în același timp.
- Concurența trebuie să fie exploatabilă:
  - Structurarea codul pentru a expune și a exploata ulterior concurenta și permiterea subproblemelor să funcționeze efectiv concomitent;
  - Majoritatea problemelor de calcul mari conțin concurență exploatabilă.
  - Un programator lucrează cu concurența exploatabilă prin crearea unui algoritm paralel și implementarea algoritmului folosind un mediu de programare paralel.
- De exemplu, să presupunem că o parte a unui calcul implică calcularea însumării unui set mare de valori.
  - Dacă sunt disponibile mai multe procesoare, în loc să adăugați valorile împreună secvențial, setul poate fi partiționat și însumările subseturilor să fie calculate simultan, fiecare pe un procesor diferit.
  - Dacă fiecare procesor are propria sa memorie, împărțirea datelor între procesoare poate permite gestionarea probls mai mari decât ar putea fi gestionate pe un singur procesor.

Probleme

- Adesea, sarcinile concomitente care constituie problema includ dependențe care trebuie identificate și gestionate corect.
  - Ordinea în care se execută sarcinile poate modifica răspunsurile calculelor în mod nedeterminist.
  - De exemplu: în însumarea paralelă descrisă anterior, o sumă parțială nu poate fi combinată cu altele decât până când calculul propriu nu va fi finalizat.
    - Algoritmul impune o ordonare parțială a sarcinilor (adică trebuie să finalizeze înainte ca sumele să poată fi combinate).
    - Valoarea numerică a sumelor se poate modifica ușor în funcție de ordinea operațiunilor din sume, deoarece aritmetica in virgula mobila nu este asociativă.
  - Un bun programator paralel trebuie să aibă grijă să se asigure că problemele nedeterministe precum acestea nu afectează calitatea răspunsului final.
- Chiar și atunci când un program paralel este „corect”, este posibil să nu furnizeze îmbunătățirea anticipată a performanței din exploatarea concurentei.
  - Trebuie urmarit ca surplusul legat de gestionarea concurentei să nu depășească timpul de rulare al programului.
  - Împărțirea activității între procesoare într-un mod echilibrat nu este adesea atât de ușor cum sugerează exemplul însumării.
- Eficiența unui algoritm paralel depinde de cât de bine se mapează pe computerul paralel de bază,
  - un alg. paralel ar putea fi eficient pe o arhitectură paralelă și un dezastru pe alta.

Concurență în programe paralele vs. sisteme de operare

- Sistemele de operare moderne utilizează mai multe procese pentru a crește randamentul sistemului
  - Conceptele fundamentale pentru manipularea în condiții de siguranță a concurenței sunt aceleași în programele paralele și în sistemele de operare.
- Diferențe importante:
  - De ce concurență
    - Într-un sistem de operare, concurența este inerentă modului în care funcționează sistemul de operare în gestionarea unei colecții de proceselor și furnizarea de mecanisme de sincronizare, astfel încât resursele să poată fi partajate în siguranță
      - Procesele nu ar trebui să poată interfera între ele,
      - Întregul sistem nu ar trebui să se prăbușească dacă ceva nu merge bine cu un singur proces.
    - În programul paralel, găsirea și exploatarea concurentei este o provocare
      - izolarea proceselor unul de celălalt nu este preocuparea critică.
  - Obiectivele de performanță sunt, de asemenea, diferite.
    - Într-un sistem de operare, este legat de randament sau de timp de răspuns și poate fi acceptat să sacrificăm eficiența pentru a menține robustetea și corectitudinea în alocarea resurselor
    - Într-un program paralel, este acela de a reduce timpul de rulare.

Ce este concurența (concomitent, simultan)

- Două evenimente sunt concurente dacă apar în același interval de timp.
  - Două sau mai multe sarcini care se execută pe același interval de timp se execută concomitent.
- Exemplul 1: Două sarcini pot apărea concomitent în aceeași secundă, dar fiecare sarcină se execută în fracțiuni diferite ale celei de-a doua.
  - Prima sarcină se poate executa pentru prima zecime din a doua și poate face o pauză,
  - A doua sarcină se poate executa pentru următoarea a zecea parte a celei de-a doua și poate face o pauză, prima sarcină poate începe să execute din nou în a treia zecime de secundă, etc..
  - Lungimea unui sec este atât de scurtă încât pare că ambele sarcini se execută simultan.
- Exemplul 2: Două programe care efectuează o anumită sarcină în aceeași oră fac continuu progresul sarcinii în timpul acelei ore, deși pot fi sau nu executate în aceeași clipă exactă.
- Sarcinile simultane se pot executa într-un mediu unic sau multiprocesare.
  - Într-un mediu de procesare unic, sarcini concomitente există în același timp și se execută în aceeași perioadă de timp prin comutarea contextului.
  - Într-un mediu multiprocesor, dacă sunt suficiente procesoare, sarcini simultane se pot executa în aceeași clipă în aceeași perioadă de timp.
- Factorul determinant pentru ceea ce face o perioadă de timp acceptabilă pentru concurență este relativ la aplicatie.

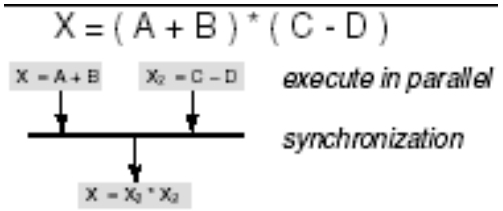
De ce concurența?

- Folosit pentru a permite unui program să lucreze mai mult în aceeași perioadă de timp sau interval de timp.
  - Programul este defalcat astfel încât unele dintre sarcini pot fi executate concomitent.
  - Uneori, are mai mult sens să ne gândim la soluția problemei ca la un set de sarcini executate simultan.
  - De exemplu, soluția problemei de a pierde în greutate este cel mai bine gândită la sarcini executate simultan: dieta și exercițiile fizice.
- Uneori, concurența este folosită pentru ca software-ul să funcționeze mai mult pe același interval în care viteza este secundară capacității.
  - De exemplu, unele site-uri web doresc ca clienții să rămână conectați cât mai mult timp. Deci nu este cât de rapid pot să-i aducă pe clienți pe site - îngrijorarea este cât de mulți clienți pe site pot suporta concomitent.
- Concurrența poate fi utilizata pentru a face software-ul mai simplu.
  - Adesea, o secvență lungă și complicată de operații poate fi implementată mai ușor ca o serie de operațiuni mici, care se execută concomitent.
- Indiferent dacă concurența este utilizata pentru a face software-ul mai rapid, pentru a gestiona sarcini mai mari sau pentru a simplifica soluția de programare, obiectivul principal este îmbunătățirea software-ului.

Două abordări de bază pentru realizarea concurentei

- Programare paralelă și programare distribuită
  - Sunt două paradigme de programare diferite care se intersectează uneori.
- Tehnicile de programare paralelă atribuie lucrul pe care un program trebuie să-l facă la două sau mai multe procesoare ale aceluiasi sistem,
  - Paralelismul din cadrul unui program poate fi împărțit în procese sau fire.
  - Multithreading-ul este limitat la paralelism.
- Tehnicile de programare distribuite atribuie lucrul pe care un program trebuie să îl facă la două sau mai multe procese - procesele pot exista sau nu pe același computer.
  - Părțile unui program distribuit rulează adesea pe diferite calculatoare sau cel puțin în diferite procese.
  - Programele distribuite pot fi împărțite numai în procese.
- Uneori programele paralele sunt distribuite (caz PVM), iar programarea distribuită este uneori folosită pentru a implementa paralelismul (caz MPI)
  - Nu toate programele distribuite implică paralelism.
  - Părțile unui program distribuit se pot executa în diferite instanțe și pe perioade de timp diferite
- În paralelism, partile care se execută concomitent sunt toate componente ale aceluiași program.
- În programele distribuite, partile sunt de obicei implementate ca programe separate.

Bazele in software ale concurentei



1. *Nivel instructiune*
   - când mai multe părți ale unei singure instrucțiuni pot fi executate simultan.
   - Figura arată cum se poate descompune o singură instrucțiune pentru executare simultană: componenta (A + B) poate fi executată în același timp cu (C - D).
   - Acest tip de paralelism este susținut în mod normal de directivele compilatorului și nu se află sub controlul direct al unui programator.
2. *Nivel rutina (functie/procedura)*
   - Structura unui program poate fi distribuită pe linii de funcții:
     - Lucrul total implicat într-o soluție software este împărțit între un număr de funcții.
   - Dacă aceste funcții sunt atribuite thread-urilor, fiecare funcție se poate executa pe un procesor diferit și dacă sunt disponibile suficiente procesoare, fiecare funcție se poate executa simultan.
3. *Nivel obiect*
4. *Nivel aplicatie*

Bazele in software ale concurentei

1. *Nivel instructiune*
2. *Nivel rutina (functie/procedura)*
3. *Nivel obiect*
   - Ideea este să avem o distribuție între obiecte.
   - Fiecare obiect poate fi atribuit unui thread sau proces diferit.
   - Obiectele aflate în diferite thread-uri sau procese pot executa simultan metodele lor.
4. *Nivel aplicatie*
   - Două sau mai multe aplicații pot coopera împreună pentru a rezolva o problemă.
   - Deși inițial cererea a fost proiectată separat și în scopuri diferite, principiile reutilizării codului permit adesea aplicațiilor să coopereze.
   - În aceste condiții, două aplicații separate lucrează împreună ca o singură aplicație distribuită.

Provocări în procesul de proiectare

- Există 3 provocări de bază pentru a scrie programe paralele:
  1. Identificarea paralelismului natural care apare în contextul domeniului problemei.
  2. Împărțirea software-ului în mod corespunzător în două sau mai multe sarcini care pot fi realizate în același timp pentru a realiza paralelismul necesar.
  3. Coordonarea acelor sarcini, astfel încât software-ul să facă corect și eficient ceea ce se presupune că va face.
- Procesul de proiectare include trei probleme (DCS):
  (a) D: descompunere,
  (b) C: comunicare, and
  (c) S: sincronizare.

- Obstacole:
  - Cursa pentru date,
  - Detectarea unui impas,
  - Eșec parțial,
  - Latență,
  - Eșecuri în comunicare,
  - Detectarea terminării,
  - Lipsa cunoasterii stării globale,
  - Ceasuri multiple,
  - Erori localizate,
  - Lipsa alocării centralizate a resurselor

  …

Descompunere

- Este procesul de împărțire a problemei și a soluției în părți.
- Părțile sunt grupate:
  - în zone logice (de exemplu, căutare, sortare, calcul, intrare, ieșire etc.).
  - prin resurse logice (fișier, comunicare, imprimantă, bază de date etc.).
- WBS (structura defalcării lucrului) determină ce parte din software ce face.
  - Nu există o abordare simplă/ca la cartea de bucate pentru identificarea WBS.
  - Procesul de modelare descoperă WBS-ul unei soluții software.
- La intrebarea cu privire la modul de împărțire a unei aplicații în execuție simultană a pieselor ar trebui să se răspundă în faza de proiectare și ar trebui să fie evidentă în modelul soluției.
  - Dacă modelul problemei și soluția nu implică sau sugerează paralelism și distribuție, încercați o soluție secvențială.
  - Dacă soluția secvențială nu reușește, acel eșec poate oferi indicii despre abordarea paralelismului.

Comunicare si sincronizare

- Odată ce soluția software este descompusă într-un număr de piese care execută concomitent, aceste părți vor avea o anumită cantitate de comunicare. Dar
  - Cum se va efectua această comunicare dacă piesele sunt în diferite procese sau computere diferite?
  - Diferitele părți trebuie să partajeze orice memorie?
  - Cum va ști o parte din software când se termină cealaltă parte?
  - Care parte începe mai întâi?
  - Cum va ști o componentă dacă o altă componentă a eșuat?
- Componentele software-ului lucrează la aceeași problemă => trebuie sincronizate. Dar
  - Toate părțile încep în același timp sau unele funcționează în timp ce altele așteaptă?
  - Ce două sau mai multe componente au nevoie de acces la aceeași resursă?
  - Cine o primește primul?
  - Dacă unele dintre părți își termină munca cu mult înainte de altele, ar trebui să li se atribuie o muncă nouă?
  - Cine atribuie noua lucrare în astfel de cazuri?
- DCS (descompunere, comunicare și sincronizare) este minimul care trebuie luat în considerare la abordarea programării paralele. Dar
  - Pe lângă faptul că are în vedere DCS, locația DCS este, de asemenea, importantă.
  - Există mai multe straturi de concordanță în dezvoltarea aplicațiilor: DCS se aplică puțin diferit în fiecare strat.

Sarcina, procesele și procesoarele

- *O sarcină* (task) este o piesă definită în mod arbitrar a unui program si este cea mai mică unitate de concurență pe care o poate exploata programul paralel
  - o sarcină individuală este executată de un singur procesor și concurența este exploatată în cadrul tuturor sarcinilor.
  - Exemplu: în Raytrace o rază sau un grup de raze și în Data Mining se poate verifica o singură tranzacție pentru apariția unui set de articole.
  - Ceea ce constituie exact o sarcină nu este prescris de programul secvențial de bază; este o alegere a agentului de paralelizare, deși se potrivește de obicei cu o oarecare granularitate naturală a lucrării în structura secvențială a programului.
  - Dacă cantitatea de muncă pe care o realizează o sarcină este mică, se numește sarcină cu granulație fină; în caz contrar, se numește granulat grosier.
- *Un proces* (denumit în continuare interschimbabil ca thread) este o entitate abstractă care execută sarcini.
  - Un program paralel este compus din mai multe procese cooperante, fiecare realizând un subset de sarcini din program.
  - Sarcinile sunt atribuite proceselor printr-un mecanism de atribuire.
  - Exemplu: în extragerea datelor, atribuirea poate fi determinată de ce porțiuni ale bazei de date sunt alocate fiecărui proces și de modul în care seturile de articole dintr-o listă de candidați sunt alocate proceselor pentru a căuta baza de date.
  - Este posibil ca procesele să fie necesare să comunice și să se sincronizeze între ele pentru a-și îndeplini sarcinile atribuite.
  - Modul în care procesele își îndeplinesc sarcinile atribuite este prin executia lor pe procesoarele fizice din mașină.

Procese vs. procesoare

- Procesoarele sunt resurse fizice.
- Procesele oferă un mod convenabil de a abstractiza sau virtualiza un multiprocesor.
- Se scriu programe paralele în termeni de procesele, nu procesoare fizice;
- Maparea proceselor către procesoare este un pas ulterior.
- Numărul de procese nu trebuie să fie același cu numărul de procesoare disponibile programului.
  - Dacă există mai multe procese, acestea sunt multiplexate pe procesoarele disponibile;
  - Dacă există mai puține procese, atunci unele procesoare vor rămâne inactive.
- Definiția corectă a sistemului de operare a unui proces este cea a unui spațiu de adrese și a unuia sau mai multor fire de control care împărtășesc acel spațiu de adrese.
  - Astfel, în această definiție se disting procesele și firele.
  - În ceea ce urmează: presupunem că un proces are un singur fir de control.

Cei patru pași și obiectivele lor

Crearea unui program paralel dintr-unul secvențial constă în 4 pași:

1. **Decompunerea** calculului în sarcini,
2. **Asignarea** de sarcini la procese,
3. **Orchestrarea** accesului la date, comunicare și sincronizare între procese,
4. **Maparea** sau legarea proceselor la procesoare.

Descompunere + asignare = **partitionare**

- întrucât împart munca depusă de program între procesele cooperante.

Pas	**Dependent de arhitectura?**	**Obiective majore de performanță**
Descompunere	Nu, in general	Expuneti o concurenta suficienta, dar nu prea mult
Asignare	Nu, in general	Balansarea incarcarii
		Reducerea volumului comunicării
Orchestrare	Da	Reducerea comunicarea non-inerentă. prin localitatea datelor
		Reducerea costului comunicării / sincronizării la procesor
		Reducerea serializarii resurselor partajate
		Planificarea sarcinile pentru a satisface dependențele in timp
Maparea	Da	Se pun procesele conexe pe același procesor dacă e necesar
		Exploatarea localității în topologia rețelei

Descompunere

- Inseamna despartirea calculului intr-o colectie de sarcini.
- Sarcinile pot deveni disponibile dinamic pe măsură ce programul se execută,
  - numărul de sarcini disponibile la un moment dat poate varia în funcție de execuția programului.
  - Numărul maxim de sarcini disponibile la un moment dat oferă o limită superioară a numărului de procese (& procesoare) care pot fi utilizate în mod eficient în acel moment.
- Scopul principal în descompunere este de a expune suficientă concurență pentru a menține procesele ocupate în permanență,
  - Concurența limitată este cea mai fundamentală limitare a accelerării realizabile prin paralelism
- Profilul de concurență:
  - având în vedere o descompunere și o dimensiune a problemei, un profil de concurență descrie câte operații (sau sarcini) sunt disponibile pentru a fi efectuate simultan în aplicație la un moment dat.
  - Este o funcție de problema, descompunere și dimensiunea problemei.
  - Este independent de numărul de procesoare, de asignare si orchestrare.
  - Pot fi ușor de analizat sau pot fi destul de neregulate.

Exemplu

- Fie exemplul unui program simplu cu două faze.
  - Faza 1: o operație este efectuată independent pe toate punctele unei grille 2-d n x n .
  - Faza 2: se calculeaza suma celor n^2 valori.
- Solutia 1:
  - Daca se vor folosi p procesoare, se pot asigna n^2/p puncta la fiecare si complete faza 1 in timp n^2/p.
  - In faza 2, fiecare procesor poate adăuga valorile alocate într-o sumă globală.
  - Care este problema si cum putem expune mai multa concurenta?
    - Problema este că acumulările în suma globală trebuie să fie făcute pe rând, sau să fie serializate, pentru a evita corupția valorii sumei cand 2 procese încearcă să o modifice simultan.
    - Astfel, a doua fază este eficient serială și durează un timp n^2 indiferent de p .
    - Timpul total este is $n^2/p+ n^2$ comparat cu timpul secvential $2n^2$, astfel S este cel mult 2 chiar daca p creste
- Solutia 2:
  - În loc să însumăm fiecare valoare direct în suma globală, să serializăm toată însumarea, împărțim a doua fază în două faze.
    - În noua fază a doua (complet paralelă), un proces însumează valorile atribuite în mod independent într-o sumă privată.
    - Apoi, în a treia fază (serializate), procesele adună sumele lor private în suma globală.
    - Timpul total este $n^2/p+ n^2/p+p$,si acceleratia este imbunatatita.
    - Limita de accelerare este aproape liniară în numărul de procesoare utilizate.

Asignare

- Precizeaza mecanismul prin care sarcinile vor fi distribuite între procese.
- Obiectivele principale ale performanței sunt:
  - Balansarea incarcarii intre procese,
    - incarcarea include: calcul, intrare / ieșire și acces la date sau comunicare,
  - Reducerea comunicarii intre procese,
  - Reducerea surplusului datorat asignarii.
- Programele sunt adesea structurate în faze, iar sarcinile candidaților pentru descompunerea în cadrul unei faze sunt adesea identificate cu ușurință.
  - Alocarea corespunzătoare a sarcinilor este adesea identificată fie prin inspecția codului, fie printr-o înțelegere a aplicației la un nivel superior.
  - Și acolo unde nu este așa, sunt adesea aplicate tehnici binecunoscute euristice.
- Clasificare:
  - *Asignare statică sau predeterminată*;
    - Dacă asignarea este complet determinată la începutul programului - sau imediat după citirea și analizarea intrării - și nu se schimbă ulterior.
  - *Asignare dinamică*:
    - dacă alocarea lucrărilor la procese este determinată la runtime pe măsură ce programul se execută - poate să reacționeze la dezechilibrele de încărcare.

Orchestrare

- Orchestrarea folosește mecanismele disponibile pentru îndeplinirea obiectivelor:
  - Pentru a-și executa sarcinile atribuite, procesele au nevoie de mecanisme pentru a denumi și accesa date, pentru a schimba date cu alte procese și pentru a se sincroniza unul cu altul.
- Opțiunile făcute în orchestrare depind mult mai mult:
  - modelul de programare,
  - eficiențele cu care sunt acceptate primitivele modelului de programare & communicare,

  decât alegerile făcute în etapele anterioare.
- Unele întrebări din orchestrare includ:
  - modul de organizare a structurilor de date și de programare a sarcinilor pt. exploatarea localității,
  - comunicare implicita sau explicita și în mesaje mici sau mari,
  - cum se organizează și exprimă exact comunicarea și sincronizarea interproceselor.
- Orchestrarea include, de asemenea, programarea sarcinilor atribuite unui proces temporal, adică decizia ordinii în care sunt executate.
- Principalele obiective de performanță în orchestrare sunt:
  - *reducerea costului comunicării și sincronizării așa cum se vede de procesoare*,
  - *păstrarea localității de referință a datelor*,
  - *programarea sarcinilor a.î. cele de care depind multe alte sarcini să fie finalizate din timp*,
  - *reducerea surplusului managementului paralelismului*.

Mapare

- Programul poate controla maparea proceselor către procesoare, dar dacă nu, sistemul de operare va avea grijă de acesta, oferind o execuție paralelă.
  - Maparea tinde să fie destul de specifică sistemului sau mediului de programare.
- In cazul partajarii spatiului
  - procesoarele din mașină sunt partiționate în subseturi fixe, eventual întreaga mașină și numai un singur program rulează la un moment dat într-un subset.
  - Programul poate lega sau fixa procesele de procesoare pt. a se asigura că acestea nu migrează în timpul execuției sau poate controla ce procesor reulează un proces pentru a păstra localitatea de comunicare în topologia rețelei
  - In multiprocessor.
- La cealaltă extremă: sistemul de operare poate controla în mod dinamic ce proces se execută unde și când - fără a permite utilizatorului niciun control asupra mapării - pentru a obține o mai bună partajare și utilizare a resurselor.
  - Fiecare procesor poate utiliza criteriile obișnuite de planificare multi-programate pentru a gestiona procesele din aceleași sau din diferite programe, iar procesele pot fi mutate între procesoare așa cum dictează programatorul.
  - Sistemul de operare poate extinde criteriile de planificare a uniprocesorului pentru a include probleme specifice multiprocesorului.
- De fapt, majoritatea sistemelor moderne se situează undeva între cele două extreme:
  - utilizatorul poate solicita sistemului să păstreze anumite proprietăți,
  - oferind programului utilizator un anumit control asupra mapării,
  - sistemul de operare este permis să modifice maparea dinamic pentru o gestionare eficientă a resurselor.

Paralelizarea calculului versus date

- *Vedere centrată pe calcul*:
  - Procesului de paralelizare este, de obicei, centrat pe calcul, mai degrabă decât pe date.
  - Este calculul care este descompus și atribuit.
  - Programatorul este responsabili pentru descompunerea și alocarea datelor și proceselor.
- În multe clase importante de probleme, descompunerea lucrărilor și a datelor sunt atât de strâns legate încât sunt dificile sau chiar inutile de distins.
  - Exemplu: data mining,
    - Procesul căruia i se atribuie o porțiune de date va fi apoi responsabil pentru calculul asociat cu acea porțiune, un așa-numit aranjamen proprietarul calculează.
  - Mai multe sisteme de limbaj, inclusiv HPF, permit programatorului să specifice descompunerea și alocarea structurilor de date;
- Distincția dintre calcul și date este mai puternică în multe alte aplicații
  - Exemplu: calculele din Raytracer

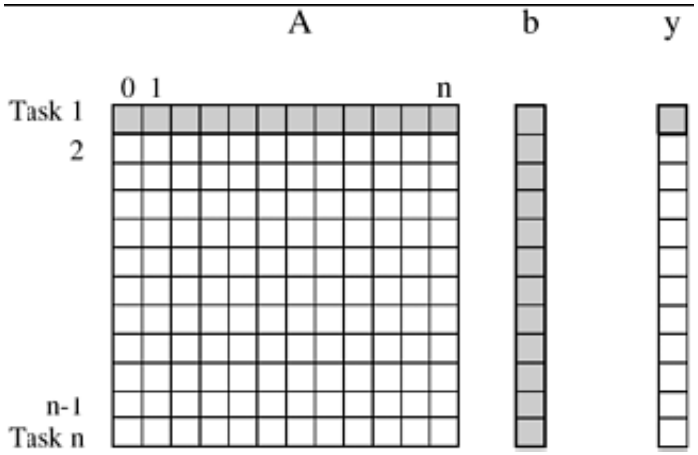
Sarcini in faza de descompunere

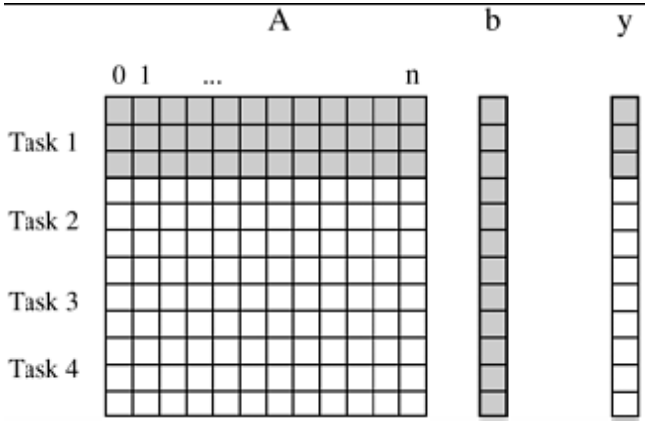
- Sarcinile sunt unități de calcul definite de programator în care calculul principal este subdivizat prin descompunere.
- Executarea simultană a mai multor sarcini este cheia reducerii timpului necesar rezolvării întregii probleme.
- Sarcinile pot avea dimensiuni arbitrare, dar odată definite, ele sunt considerate unități de calcul indivizibile.
- Sarcinile în care se descompune o problemă nu au toate aceeasi dimensiune.
- Exemple care urmează:

1. Înmulțire matrice densă-vector
2. Înmulțire matrice rară-vector

Ex 1: Înmulțire matrice densă-vector - descompunere

- Luați în considerare înmulțirea unei matrice *A* dense *n* x *n* cu un vector *b* pentru a produce un vector *y*.
  - Elementul *i*th, *y [i],* al vectorului produs este produsul punct al liniei *i* al A cu vectorul de intrare *b*; adica *y*[*i*]=sum(*A*[*i,j*]b[*j*],*j*=1..*n*).
  - Calculul fiecărui *y [i]* poate fi considerat ca o sarcină.
    - toate sarcinile sunt independente și pot fi îndeplinite toate împreună sau în orice secvență.
  - Alternativ:
    - calculul ar putea fi descompus în mai puține, să spunem 4, sarcini în care fiecare sarcină calculează aproximativ *n* / 4 din intrările vectorului *y*.





Granularitate și interacțiunile sarcinilor

- Numărul și dimensiunea sarcinilor în care este descompusă o problemă determină granularitatea descompunerii.
  - O descompunere într-un număr mare de sarcini mici se numește cu granulație fină;
  - O descompunere într-un număr mic de sarcini mari se numește granulație grosieră.
- Exemple:
  - Prima descompunere pentru înmulțirea matrice-vector este cu granulație fină.
  - A doua este cu granulație grosieră.
- *Gradul maxim de concurență*:
  - Nr. maxim de sarcini executate simultan într-un program paralel la un moment dat.
  - De obicei mai mic decât nr. total de sarcini datorate dependențelor dintre sarcini.
  - Ex.: pentru graficele de dependență de sarcini tip arbori, e nr. frunze din arbore.
- Gradul de concurenta mediu,
  - numărul mediu de sarcini care pot fi executate concomitent pe toată durata de execuție a programului.
- Exemplu:
  - Prima descompunere a înmulțirii matrice-vector are o granularitate destul de mică și un grad mare de concurență.
  - A doua descompunere pentru aceeași problemă are o granularitate mai mare și un grad mai mic de concurență.

Graful dependenței sarcinilor

- Unele sarcini pot utiliza datele produse de alte sarcini și, astfel, poate fi necesar să aștepte ca aceste sarcini să termine execuția.
- Este o abstractizare folosită pentru a exprima astfel de dependențe între sarcini și ordinea lor relativă de execuție.
- Un graf de dependență a sarcinilor este un graf aciclic direcționat în care nodurile reprezintă sarcini, iar arcele direcționate indică dependențele dintre ele.
  - Sarcina corespunzătoare unui nod poate fi executată atunci când toate sarcinile conectate la acest nod prin margini de intrare s-au finalizat.
  - Graful de dependență poate fi deconectate și setul de arce poate fi gol.
    - De exemplu in inmultirea matrice-vector.
- Nodurile fără arce de intrare sunt noduri de pornire și nodurile fără arce de ieșire sunt noduri de finisare.
- Gradul de concurență depinde de forma grafului de dependență a sarcinilor.
- O caracteristică a unui graf de dependență de sarcină care determină gradul mediu de concurență pentru o granularitate dată este drumul critic.
  - Cel mai lung drum direcțional între orice pereche de noduri de început și terminare.
  - Suma greutăților nodurilor de-a lungul acestei căi este lungimea critică a drumului, unde greutatea unui nod este mărimea sau cantitatea de lucru asociată sarcinii corespondente.
  - Raportul dintre cantitatea totală de lucru și lungimea drumului critic: grad mediu de concurenta.
  - O cale critică mai scurtă favorizează un grad mai mare de concurentă.

Factori de limitare

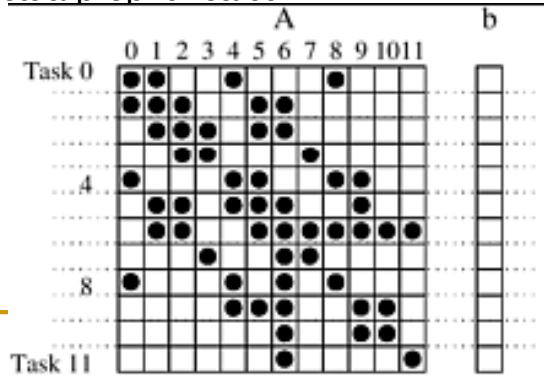
- De obicei, există o limită inerentă cu privire la modul în care o descompunere cu granulație fină este permisa.
  - De exemplu, sunt n^2 înmulțiri și adunări în înmulțirea matrice-vector considerată în exemplu și problema nu poate fi descompusă în mai mult de $O(n^2)$ sarcini chiar și folosind cea mai fină descompunere.
- Interacțiunea dintre sarcinile care rulează pe diferite procesoare fizice.
  - Sarcinile in căre o problemă este descompusă partajeaza adesea intrarile, ieșirile sau datele intermediare.
  - Dependențele dintr-un graf de dependență ale sarcinilor rezultă de obicei din faptul că ieșirea unei sarcini este intrarea pentru alta.
  - Exemplu: în exemplul de interogare a bazei de date, sarcinile partajează date intermediare; tabelul generat de o sarcină este adesea folosit de o altă sarcină ca input.

Graful de interactiune a sarcinilor

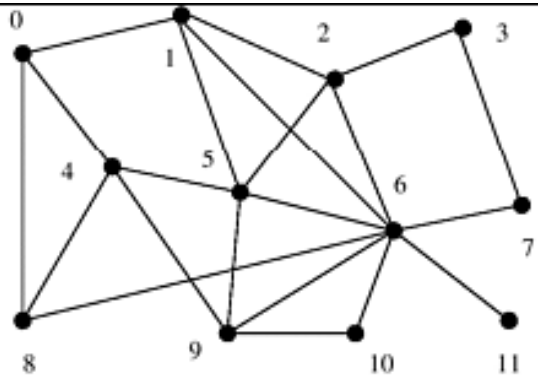
- În funcție de definiția sarcinilor și de paradigma de programare paralelă, pot exista interacțiuni între sarcini care par independente într-un graf de dependență a sarcinilor.
  - De exemplu, în descompunerea pentru înmulțirea matrice-vector, deși toate sarcinile sunt independente, toate au nevoie de acces la întregul vector de intrare *b*.
    - Întrucât inițial există o singură copie a lui *b*, sarcinile pot fi obligate să trimită și să primească mesaje pentru ca toți să poată accesa întregul vector în paradigma cu memorie distribuită.
- Graful de interactiune a sarcinilor:
  - Captează sablonul de interacțiune între sarcini,
  - Nodurile dintr-un grafic de interacțiune sarcină reprezintă sarcini, iar arcele conectează sarcini care interacționează între ele.
  - Nodurilor și arcelor le pot fi atribuite greutăți proporționale cu cantitatea de calcul realizată de o sarcină și cantitatea de interacțiune care are loc de-a lungul unei margini, dacă aceste informații sunt cunoscute.
  - Arcele sunt de obicei nedirecționate, dar arcele direcționate pot fi utilizate pentru a indica direcția de curgere a datelor, dacă este un graf unidirecțional.
  - Multimea arcelor unui grafic de interacțiune a sarcinilor este, de obicei, o supramultime a multimii de arce ale grafului de dependență a sarcinilor.
  - Exemplu: în exemplul de interogare a bazei de date, graful de interacțiune a sarcinilor este același cu graful de dependență a sarcinilor.

Ex. 2: Inmultirea matrice rara - vector

- Fie: ? $y = Ab$ a matricei rare n x n A cu vectorul dens n dimensional b.
  - O matrice este considerată rară atunci când un nr semnificativ. intrările din acesta sunt zero și locațiile intrărilor non-zero nu se conformează unei structuri sau sablon predefinite.
  - Operațiunile aritmetice care implică matrici rare pot fi adesea optimizate semnificativ prin evitarea calculelor care implică zerouri.
- Cand se calculeaza a i-a intrare $y[i]$=sum($A[i,j]b[j]$,j=1..n) a vectorului produs, se calculeaza produsele $A[i, j]$ x $b[j]$ numai pentru acele valori j pt. care $A[i, j]$ nu este 0.
  - De exemplu, $y[0] = A[0, 0].b[0] + A[0, 1].b[1] + A[0, 4].b[4] + A[0, 8].b[8]$.
- Un mod posibil de descompunere a acestui calcul este să impartirea vectorul y a.i. o sarcină calculeaza o intrare în ea (Fig. (a))
  - Alocarea calculului elementului y [i] al vectorului de ieșire la Task i,
  - Sarcina i este „proprietarul” rândului A[i,*] al matricei și al elementului b[i].
  - Calculul y[i] necesită acces la multe elemente ale lui b care sunt deținute de alte sarcini.
  - Deci Task i trebuie să obțin aceste elemente din locațiile corespunzătoare.
    - În paradigma de transmitere a mesajelor, cu proprietatea lui b[i], Task i moștenește, de asemenea, responsabilitatea de a trimite b [i] la toate celelalte sarcini care au nevoie pentru calcularea lor.
    - De exemplu: sarcina 4 trebuie să trimită b [4] la sarcinile 0, 5, 8 și 9 și trebuie să obțină b [0], b [5], b [8] și b [9] pentru a-și efectua propriul calcul.
- Graful de interacțiune a sarcinilor este prezentat înFig.(b).



(a)



(b)

Clasificarea tehnicilor de descompunere

1. Descompunere recursiva,
2. Descompunerea datelor,
3. Descompunerea exploratorie,
4. Descompunerea speculativa

- Tehnicile recursive și de descompunere a datelor au un scop relativ general, deoarece pot fi folosite pentru a descompune o mare varietate de probleme.
- Tehnicile de descompunere speculativă și exploratorie sunt mai mult cu un scop special, deoarece se aplică la clase specifice de probleme.