X. Tenhici de mapare

Continut

- clasificarea maparii
- scheme de mapare statica
- scheme de mapare dinamica
- maximizarea localitatii datelor
- suprapunera calculelor cu comunicarea,
- replicarea,
- optimizarea interactiunilor colective

Sarcini vs. procese vs. procesoare

- Procesoare
  - sunt unitățile hardware care efectuează fizic calcule.
  - în majoritatea cazurilor, există o corespondență unu la unu între procese și procesoare.
- Sarcinile, în care o problemă este descompusă, se execută pe procesoare fizice.
- Un proces:
  - se referă la un agent de procesare sau de calcul care execută sarcini.
  - este o entitate abstractă care folosește codul și datele corespunzătoare unei sarcini pentru a produce ieșirea acelei sarcini într-un interval finit de timp după ce sarcina este activată de programul paralel.
  - Pe lângă efectuarea calculelor, un proces se poate sincroniza sau comunica cu alte procese, dacă este necesar.
  - Pentru a obține orice accelerare peste o implementare secvențială, un program paralel trebuie să aibă mai multe procese active simultan, care lucrează la diferite sarcini.

O mapare bună(1)

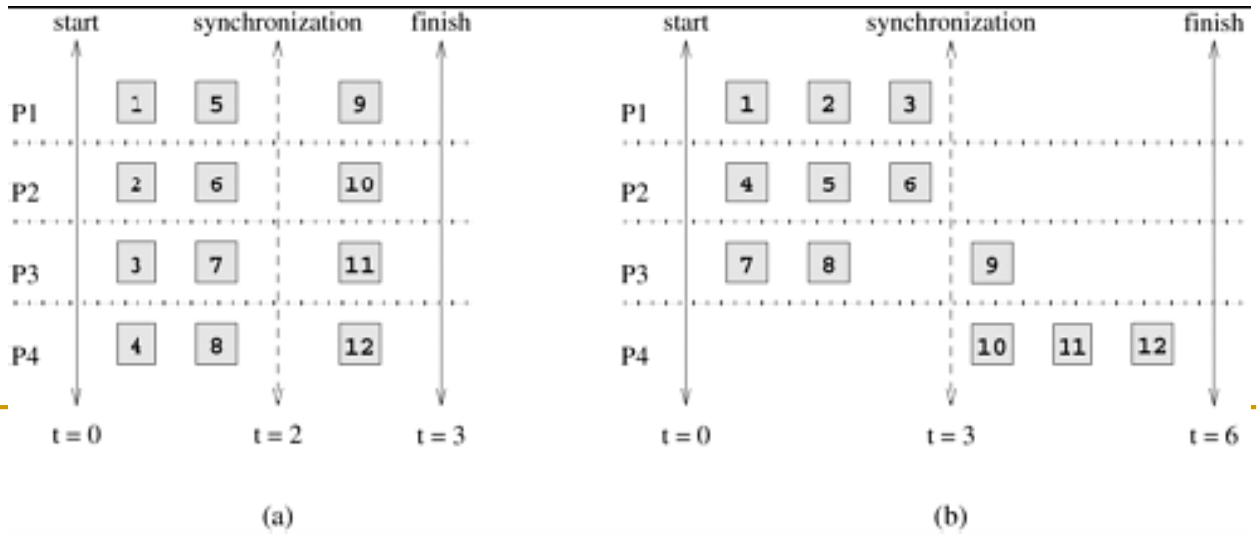
- Mecanism prin care sarcinile sunt atribuite proceselor pentru execuție.
- Grafurile de dependență ale sarcinilor și de interacțiune ale sarcinilor care rezultă dintr-o alegere a descompunerii joacă un rol important în selecția unei bune mapări.
- O mapare bună ar trebui să caute:
  1. Maximizeze utilizarea concurgenței prin maparea sarcinilor independente pe diferite procese,
  2. Minimizeze timpul total de finalizare asigurându-se că procesele sunt disponibile pentru a executa sarcinile pe drumul critic imediat ce aceste sarcini devin executabile.
  3. Minimizeze interacțiunea dintre procese prin cartografierea sarcinilor cu un grad ridicat de interacțiune reciprocă cu același proces.
- În majoritatea algoritmilor paraleli nontriviali, cele trei tind să fie obiective contradictorii.
  - Exemplu: cea mai eficientă combinație de descompunere-mapare este o singură sarcină mapată pe un singur proces. Nu pierde timp în interacțiune, dar nici nu atinge nicio accelerare.
- Maparea sarcinilor pe procese joacă un rol important în determinarea cât de eficient este algoritmul paralel rezultat.
  - Chiar dacă gradul de concurgență este determinat de descompunere, maparea este determinant in cât de multă concurență este utilizată efectiv și cât de eficient.

O mapare buna (2)

- Odată ce un calcul a fost descompus în sarcini, aceste sarcini sunt mapate pe procese cu obiectul că toate sarcinile se finalizează în cea mai scurtă perioadă de timp scursă.
  - Pentru a realiza un timp scurt de execuție, surplusul executării sarcinilor în paralel trebuie să fie minim.
- Pentru o descompunere dată, există două surse cheie ale surplusului.
  1. Timpul petrecut în interacțiunea dintre proces.
  2. Timpul în care unele procese pot petrece fiind inactive; datorita:
  - Distribuția neuniformă a sarcinilor poate determina finalizarea unor procese mai devreme decât altele.
  - Toate sarcinile neterminate mapate pe un proces pot aștepta finalizarea sarcinilor mapate pe alte procese pentru a satisface constrângerile impuse de graful de dependență a sarcinilor.
- O bună mapare a sarcinilor asupra proceselor trebuie să depună eforturi pentru a atinge obiectivele reducerii:
  1. cantitatea de timp petrecută în procesele de interacțiune între ele,
  2. cantitatea totală de timp de inactivitate a proceselor.
- Aceste două obiective adesea intră în conflict între ele.
  - Exemplu:
    - Min. interacțiunilor poat fi realizat cu ușurință prin alocarea unor seturi de sarcini care trebuie să interacționeze între ele pe același proces.
    - O astfel de cartografiere va avea ca rezultat o sarcină de lucru extrem de dezechilibrată între procese.
    - Respectarea acestei strategii până la limită va mapa adesea toate sarcinile pe un singur proces.

Exemplu: Două mapări ale unei descompuneri ipotetice cu o sincronizare

- Un grafic de dependență ale sarcinilor determină ce sarcini se pot executa în paralel și care trebuie să aștepte ca unele altele să termine la o anumită etapă în execuția unui algoritm paralel.
- Sincronizarea slabă între sarcinile care interacționează poate duce la inactivitate dacă una dintre sarcini trebuie să aștepte pentru a trimite sau primi date de la o altă sarcină.
- O bună mapare asigură că calculele și interacțiunile dintre procesele din fiecare etapă de execuție a alg. paralel sunt bine echilibrate.
- Fig. Prezintă două mapări ale descompunerii cu 12 sarcini în care ultimele 4 sarcini pot fi pornite numai după terminarea primelor 8 datorită dependențelor dintre sarcini.
  - două mapări, fiecare cu o sarcină de lucru globală echilibrată, pot duce la timpi de finalizare diferiti.



Clasificarea maparilor

- Alegerea unei bune mapări în acest caz depinde de mai mulți factori, inclusiv:
  - cunoașterea dimensiunilor sarcinilor,
  - dimensiunea datelor asociate sarcinilor,
  - caracteristicile interacțiunilor între sarcini,
  - paradigma de programare paralelă.
- O mapare optima este o problemă completă NP pentru sarcini neuniforme.

1. **Mapare statica:**
   - Distribuie sarcinile între procesele anterior executării algoritmului.
   - Pentru sarcinile generate static, poate fi utilizată o mapare statică sau dinamică.
   - În multe cazuri practice, euristica relativ ieftină oferă soluții aproximative destul de acceptabile pentru problema optimă a mapării statice.
   - Algoritmii care utilizează maparea statică sunt în general ușor de proiectat și programat.
   - Dacă dimensiunile sarcinii nu sunt cunoscute,poate duce la dezechilibru grav de sarcină.
2. **Maparea dinamica:**
   - Distribuie lucrul între procese în timpul executării algoritmului.
   - Dacă sarcinile sunt generate dinamic, atunci ele trebuie mapate și dinamic.
   - Dacă cantitatea de date asociate sarcinilor este mare în raport cu calculul, atunci o mapare dinamică poate presupune mutarea acestor date între procese.
   - Într-o paradigmă cu spațiu de adresă partajat, maparea dinamică poate funcționa bine chiar și cu date mari asociate sarcinilor, dacă interacțiunea este numai de citire.
   - Algs. care necesită mapare dinamică sunt complicate, în special în paradigma de transmitere a mesajelor.

Mapare statică: mapări bazate pe partiţionarea datelor

- Mapare statică:
  - este adesea utilizata în combinație cu o descompunere bazată pe partiționarea datelor.
  - utilizat pentru anumite probleme care sunt exprimate în mod natural printr-un grafic static de dependență a sarcinilor.
- Într-o descompunere bazată pe date de partiționare, sarcinile sunt asociate îndeaproape cu porții de date de regula proprietar-calculeaza.

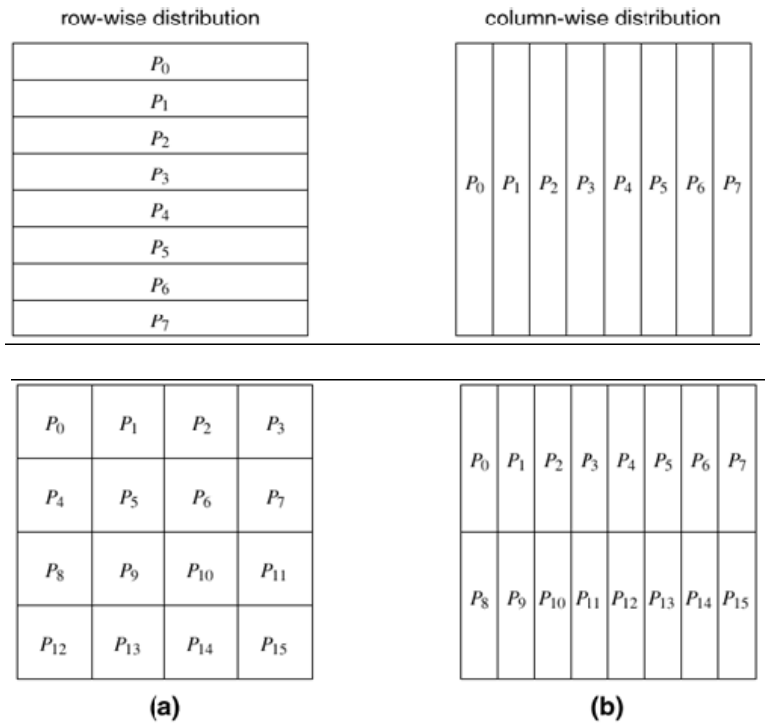
⇒ maparea datelor relevante pe procese este echivalentă cu sarcinile de mapare pe procese.

- Două dintre cele mai comune moduri de reprezentare a datelor în algoritmi:
  1. tablouri
  2. grafuri.

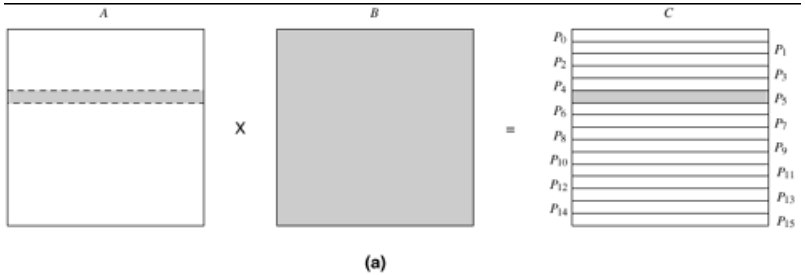
Distributii in bloc

- Un tablou dimensional d este distribuit între procese: fiecare proces primește un bloc continuu de intrări de ale tabloului de-a lungul unui subset specificat de dimensiuni ale tabloului.
- Distribuțiile bloc ale tablourilor sunt adecvate atunci când există o localitate de interacțiune, adică, calcularea unui element al unui tablou necesită alte elemente din apropiere în tablou.

- Exemplu: matrice 2-d cu n rânduri și n coloane.
  - Distribuții pe rânduri:
    - Partiți tabloul în părți p astfel încât partea kth conține rânduri kn / p ... (k + 1) n / p - 1, unde 0 <= k <p.
    - Fiecare partiție conține un bloc de n / p rânduri consecutive.
  - Distributii pe coloane:
    - Partiția A de-a lungul celei de-a doua dimensiuni, apoi fiecare partiție conține un bloc de n / p coloane consecutive.
  - Distributii bi-dimensionale
    - În loc să fie selectata o dim, sunt mai multe dimensiuni
    - Se selecteaza ambele dimensiuni și este împărțita matricea în blocuri a.î. fiecare bloc să corespundă unei secțiuni n / p1 x n / p2 a matricei, cu p = p1 x p2 numărul de procese.
    - Fig. Ilustrează două distribuții 2-d diferite, pe o grilă de proces 4 x 4 și 2x 8.

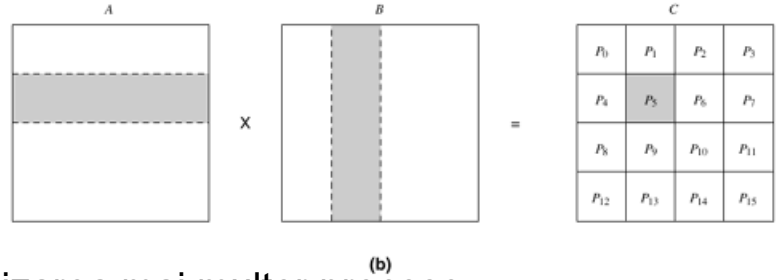


Exemplu 1: înmulțirea matricelor



- Se consideră înmulțirea matriceala n x n: C = A x B si partiționarea matrici de ieșire C.
- Echilibrarea calculele împărțind uniform C între p procese disponibile.

1. Distribuție bloc 1-d: fiecare proces va primi un bloc de n / p rânduri (sau coloane) de C,
2. Distribuția blocului în 2 d: fiecare proces va obține un bloc cu dimensiunea n / sqrt (p) x n / sqrt (p).



- Distribuțiile cu dimensiuni superioare permit utilizarea mai multor procese.

1. Distribuție bloc 1-d: până la n procese prin alocarea unui singur rând de C fiecărui proces.
2. Distribuția 2-d ne va permite să folosim până la n^2 procesează prin alocarea unui singur element de C fiecărui proces.

- Distribuțiile cu dimensiuni superioare reduc cantitatea de interacțiuni între diferitele procese pentru multe probleme.

1. Partajarea 1-d de-a lungul rândurilor, fiecare proces trebuie să acceseze n / p rândurile corespunzătoare ale matricei A și întreaga matrice B: cantitatea totală de date care trebuie accesate este $n^2/p + n^2$.
2. Distribuție 2-d: fiecare proces trebuie să acceseze n / sqrt (p) rânduri de matrice A și n / sqrt (p) coloane ale matricei B: cantitatea totală de date partajate la care trebuie să aibă acces fiecare process este $O(n^2/\text{sqrt}(p))$, << $O(n^2)$ in cazul 1-d.

Ex. 2: factorizarea LU

- Dacă cantitatea de muncă diferă pentru diferite elemente ale unei matrice, o distribuție a blocului poate duce la dezechilibre de încărcare.
- Exemplu clasic al acestui fenomen: factorizarea LU a unei matrice: cantitatea de calcul crește de la stânga sus la dreapta jos a matricei.
- Factorizarea LU determină o matrice pătrată nesingulară A ca produsul dintre:
  - o matrice triunghiulară inferioară L cu o diagonală unitară și
  - o matrice triunghiulară superioară U.
- Următorul algoritm arată algoritmul serial bazat pe coloane.
  - Matricile L și U împart spațiul cu A: A este modificat pentru a stoca L și U în părțile sale triunghiulare inferioare și superioare,

- Serial:

```
procedure COL_LU (A)
begin
for k := 1 to n do
for j := k to n do
A[j, k]:= A[j, k]/A[k, k];
endfor;
for j := k + 1 to n do
for i := k + 1 to n do
A[i, j] := A[i, j] - A[i, k] x A[k, j];
endfor;
endfor;
/* After this iteration, column A[k + 1 : n,
     k] is logically the kth
column of L and row A[k, k : n] is logically
     the kth row of U. */
endfor;
end COL_LU
```

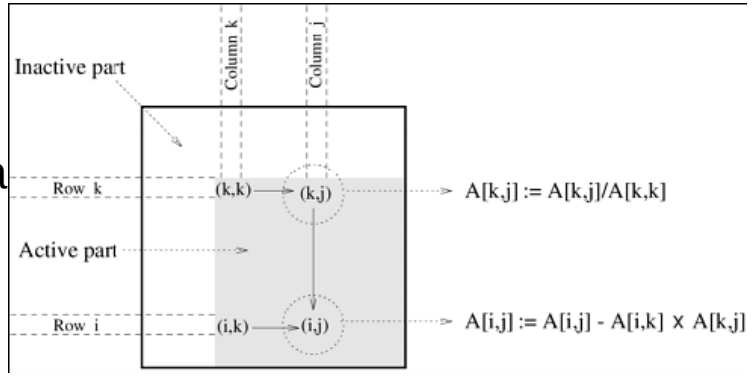
Versiune bloc a LU.

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} L_{1,1} & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix} \cdot \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ 0 & U_{2,2} & U_{2,3} \\ 0 & 0 & U_{3,3} \end{pmatrix}$$

- Fig. 1 prezintă o posibilă descompunere a factorizării LU în 14 sarcini folosind o partitionare bloc 3 x 3 a matricei și folosind o versiune bloc:
  - Pentru fiecare iterație a ciclului exterir k: = 1 la n, următorul ciclui imbricate din Alg. merge de la k + 1 la n.

1: $A_{1,1} \rightarrow L_{1,1}U_{1,1}$
2: $L_{2,1} = A_{2,1}U_{1,1}^{-1}$
3: $L_{3,1} = A_{3,1}U_{1,1}^{-1}$
4: $U_{1,2} = L_{1,1}^{-1}A_{1,2}$
5: $U_{1,3} = L_{1,1}^{-1}A_{1,3}$
6: $A_{2,2} = A_{2,2} - L_{2,1}U_{1,2}$
7: $A_{3,2} = A_{3,2} - L_{3,1}U_{1,2}$
8: $A_{2,3} = A_{2,3} - L_{2,1}U_{1,3}$
9: $A_{3,3} = A_{3,3} - L_{3,1}U_{1,3}$
10: $A_{2,2} \rightarrow L_{2,2}U_{2,2}$
11: $L_{3,2} = A_{3,2}U_{2,2}^{-1}$
12: $U_{2,3} = L_{2,2}^{-1}A_{2,3}$
13: $A_{3,3} = A_{3,3} - L_{3,2}U_{2,3}$
14: $A_{3,3} \rightarrow L_{3,3}U_{3,3}$

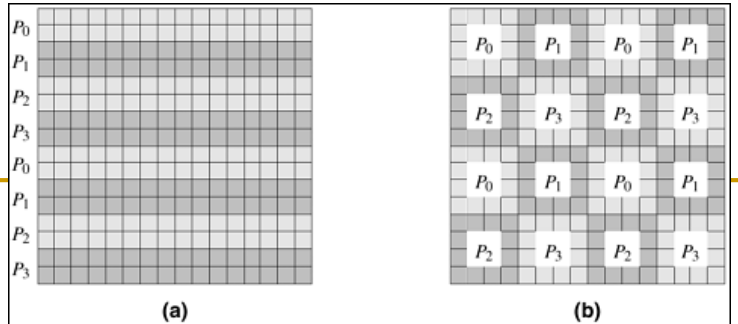
- Partea activă a matricei, așa cum se arată în Fig. 2, se micșorează spre colțul din dreapta jos al matricei pe măsură ce calculul continuă.
- Într-o distribuție în bloc, procesele alocate rândurilor și coloanelor de început ar efectua o muncă mult ma mică decât cele atribuite rândurilor și coloanelor.
- Calcularea diferitelor blocuri ale matricei necesită cantități diferite de muncă:
  - Ilustrat in Fig. 3.
  - Calculul lui A1,1 necesita o sarcina – Task 1.
  - Calculul A3,3 necesita 3 sarcini –9, 13, si 14.
- Procesul de lucru pe un bloc poate să funcționeze inactiv chiar și atunci când există sarcini neterminate asociate cu acel bloc.
  - Aceasta inactivitate poate apărea dacă constrângerile impuse de graful dependență ale sarcinilor nu permite sarcinile rămase pe acest proces să se desfășoare până când una sau mai multe sarcini mapate pe alte procese sunt finalizate.



P_0 T_1	P_3 T_4	P_6 T_5
P_1 T_2	P_4 T_6 T_{10}	P_7 T_8 T_{12}
P_2 T_3	P_5 T_7 T_{11}	P_8 T_9 T_{13} T_{14}

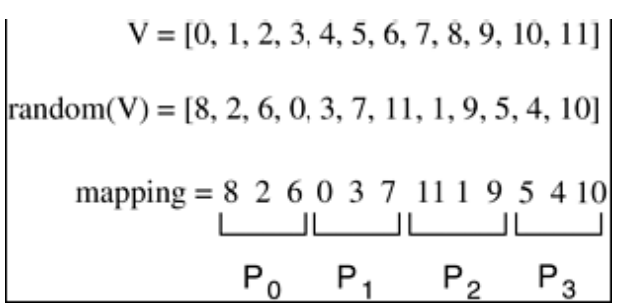
Distributie ciclica sau bloc-ciclica

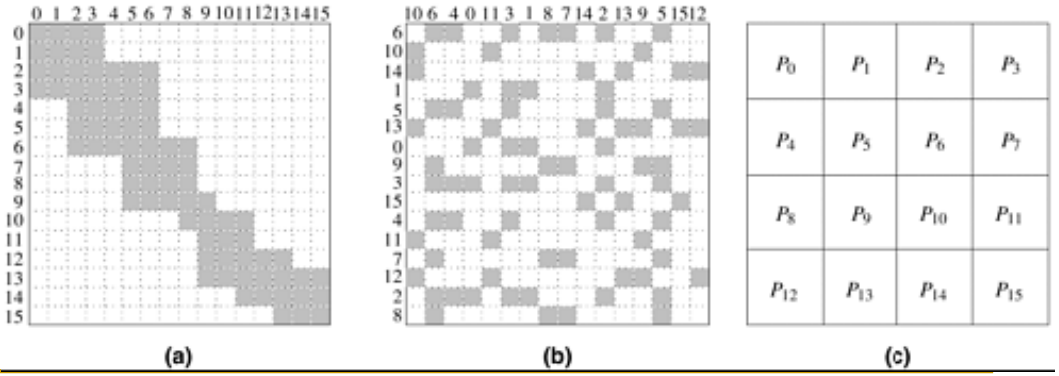
- Distribuția bloc-ciclică este o variație a schemei de distribuție bloc care poate fi folosită pentru a atenua problemele de dezechilibru și încărcare.
- Ideea centrală din spatele unei distribuții bloc-ciclice:
  - partiționează un tablou în mai multe blocuri decât numărul de procese disponibile.
  - alocarea partițiilor (și sarcinile asociate) proceselor într-o manieră rotundă, astfel încât fiecare proces să obțină mai multe blocuri care nu sunt adiacente.
- Ex. 1: o distributie 1-d bloc-ciclica a unei matrice intre *p* procese,
  - rândurile (coloanele) unui matrice *n* x *n* sunt împărțite în *αp* groupuri de *n*/(α*p*) rânduri consecutive (coloane), unde 1 <=α<=*n*/*p*.
  - Aceste blocuri sunt distribuite între *p* procesele într-un mod cuprinzător, astfel încât blocul *bi* este atribuit procesului *Pi %p* ('%' is the modulo operator).
  - Asignarea α blocuri ale matricei la fiecare proces, dar fiecare bloc care este atribuit aceluiași proces este la distanta de *p* blocuri.
- Ex. 2: o distribuție bloc-ciclică 2-d a unui tablou n x n prin partitionarea acesteia în blocuri pătrate de dimensiuni α sqrt(*p*) x α sqrt(*p*) și distribuirea lor pe o matrice de procese sqrt (p) x sqrt (p) inelat.
- Distribuția bloc-ciclică poate fi extinsă la tablouri cu dimensiuni mai mari.
- Fig ilustrează distribuțiile ciclice cu bloc 1-d și 2-d ale unui tablou bidimensional.



Distribuţii bloc aleatoare

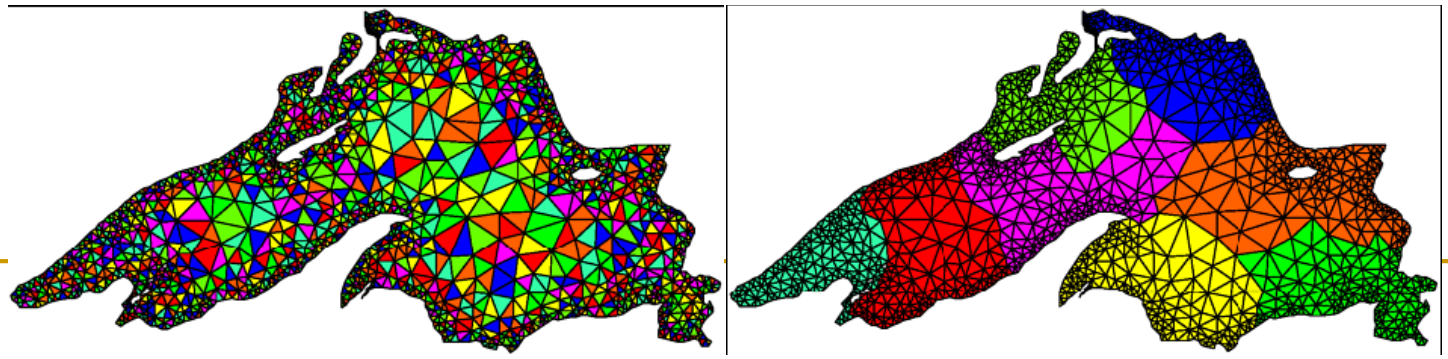
- balansarea încărcării este căutată prin împărțirea tabloului în mai multe blocuri decât numărul de procese disponibile.
- blocurile sunt distribuite uniform și aleatoriu între procese.
- Ex. 1: Ex. 2:





Partitoonarea grafurilor

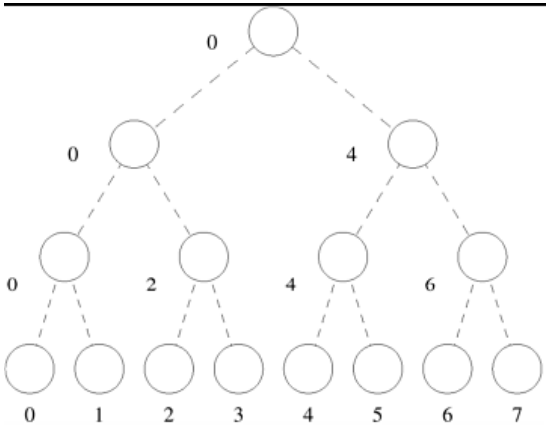
- există numeroși algoritmi care operează pe structuri de date rare și pentru care modelul de interacțiune între elementele de date este dependent de date și extrem de neregulat.
  - Simulările numerice ale fenomenelor fizice oferă o sursă mare de astfel de calcule.
    - În aceste calcule, domeniul fizic este discretizat și reprezentat de o grila de elemente.
    - Ex: Simularea unui fenomen fizic, cum ar fi dispersia unui contaminant de apă în lac implică calcularea nivelului de contaminare la fiecare vertex al acestei rețele la diverse intervale de timp.
- Aleatoriu: fiecare proces va trebui să acceseze un set mare de puncte aparținând altor procese pentru a completa calculele pentru porțiunea alocată a rețelei.
- ? Partitionarea in p parti astfel încât fiecare parte să conțină aproximativ același nr. a vârfurilor, & nr. a arcelor care traversează limitele partiției este minimizat.
  - Problema NP-completa.
  - Algoritmi care utilizează o euristică puternică sunt disponibili pentru a calcula partiții rezonabile.
  - Fiecărui proces i se atribuie o regiune contiguă a rețelei, astfel încât numărul total de puncte de rețea care trebuie accesate peste limitele partiției să fie redus la minimum.



Mapare statică: mapări bazate pe partitionarea sarcinilor

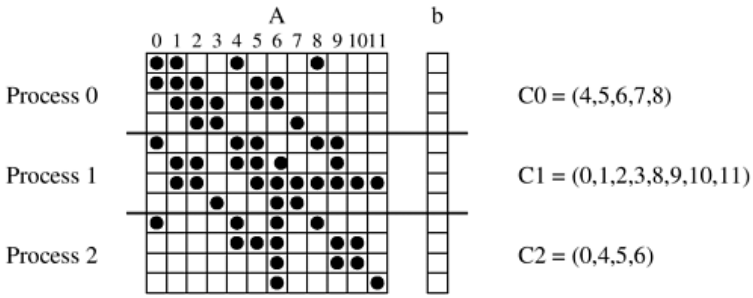
- Atunci când calculul este exprimat în mod natural sub forma unui graf static de dependență a sarcinilor cu sarcini de mărimi cunoscute.
- Obiective conflictuale de minimizare a timpului inactivitarii și minimizarea timpului de interacțiune al algoritmului paralel.
- Determinarea unei mapări optime pentru un grafic general de dependență de sarcini este o problemă completă NP

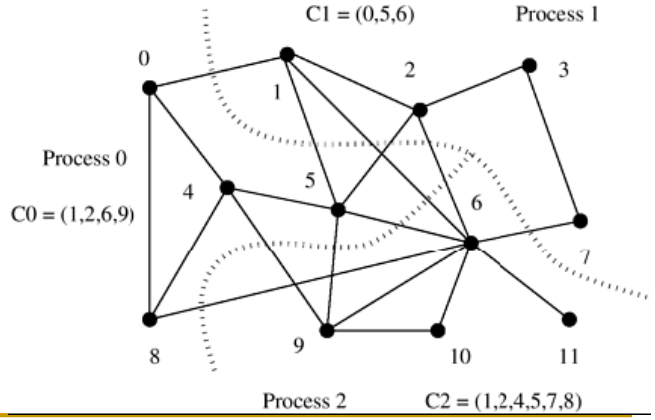
- Ex:
  - graful de dependență a sarcinilor este un arbore binar perfect (de exemplu, găsirea minimului)
  - mapare pe un hipercub
  - minimizează surplusul interacțiunii prin maparea a numeroase sarcini interdependente pe același proces (adică, sarcinile de-a lungul unei ramuri drepte a arborelui) și altele pe procese în cadrul unei legături de comunicare la o distanță 1 de cealaltă.



O solutie: partitionarea grafului de interactiune a sarcinilor

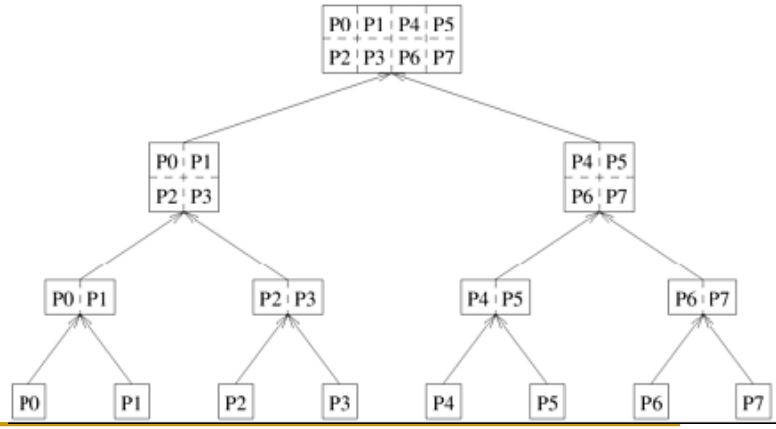
- O solutie aproximativa
- Ex: O mapare a trei pentru multiplicarea unei matrice rare cu un vector pe trei procese
  - Varianta 1: această mapare atribuie sarcini corespunzătoare a patru intrări consecutive ale b fiecărui proces.
  - Varianta 2: partiționarea pentru graful de interacțiune a sarcinilor;
    - *Ci* conține indicii lui b pe care trebuie să acționeze sarcinile de pe Procesul *i* de la sarcini mapate pe alte procese.
  - O comparație rapidă a listelor C0, C1 și C2 în cele două cazuri dezvăluie cu ușurință că a doua mapare presupune mai puține schimburi de elemente ale b între procese decât prima mapare.





Mapare statică: mapări ierarhice

- O mapare bazată exclusiv pe graful de dependență a sarcinilor poate suferi dezechilibru de încărcare sau concurență inadecvată.
- Dacă sarcinile sunt suficient de mari, atunci o mapare mai bună poate fi obținută printr-o descompunere suplimentară a sarcinilor în subsarcini mai mici.
- Ex 1: Quicksort are un graful de dependență a sarcini, care este un candidat ideal pentru o mapare ierarhică.
- Ex 2 - figura



Mapare dinamica

- Necesar in situatii
  - cand o mapare statică poate avea ca rezultat o distribuție dezechilibrată extrem de mare între procese sau
  - cand graficul de dependență a sarcinilor în sine este dinamic, împiedicând astfel o mapare statica.
- motivul principal pentru utilizarea unei mapări dinamice este echilibrarea volumului de muncă între procese,
  - maparea dinamică este adesea denumită echilibrare dinamică a sarcinii.
- Tehnicile de mapare dinamică sunt clasificate:
  - centraliza
  - distribuit.

Mapare dinamica: scheme centralizate

- toate sarcinile executabile sunt menținute într-o structură de date centrală comună sau sunt întreținute de un proces special sau un subset de procese.
- un proces special este desemnat pentru a gestiona grupul de sarcini disponibile, apoi este adesea denumit master.
- celelalte procese care depind de master pentru obținerea muncii sunt denumite sclavi
- de fiecare dată când un proces nu are nicio lucrare, este nevoie de o parte din lucrările disponibile din structura centrală de date sau din procesul master.
- de fiecare dată când este generată o nouă sarcină, aceasta este adăugată la această structură de date centralizată sau raportată la procesul master.
- schemele de echilibrare a sarcinii centralizate sunt de obicei mai ușor de implementat decât schemele distribuite, dar pot avea scalabilitate limitată.
- nr. mare de accesari la structura de date obișnuită sau procesul principal tinde să devină un blocaj.
- Exemplu:
  - problema sortării intrărilor din fiecare rând al unei matrici n x n.
  - O mapare naivă - un nr egal. rânduri la fiecare proces - pot duce la dezechilibru de încărcare.
  - O altă soluție: menținerea unui grup central de indici ai rândurilor care încă nu au fost sortate.
    - Ori de câte ori un proces este inactiv, alege un index disponibil, îl șterge și sortează rândul cu acel index
    - Planificarea iterațiilor independente ale unei bucle între procesele paralele este cunoscută sub numele de auto-programare.

Mapare dinamică: scheme distribuite

- setul de sarcini executabile sunt distribuite între procesele care schimbă sarcini în timpul rulării pentru a echilibra munca
- fiecare proces poate trimite lucru sau poate primi lucru de orice alt proces
- problemele unei scheme distribuite de echilibrare a sarcinii sunt următoarele:
  - Cum se împerechează procesele de trimitere și primire?
  - Transferul de lucru este inițiat de expeditor sau receptor?
  - Câtă muncă este transferată în fiecare schimb?
  - Când se realizează transferul de muncă?

Metode pentru reducerea surplusului interactiunii

- Este important pentru un program paralel eficient.
- Surplusul din program paralel datorat interacțiunii dintre procesele sale depinde de mulți factori:
  - volumul de date schimbate în timpul interacțiunilor,
  - frecvența interacțiunii,
  - modelul spațial și temporal al interacțiunilor, etc.
- Unele tehnici generale care pot fi utilizate pentru a reduce surplusul interacțiunii suportate de programele paralele:
  1. Maximizarea localității datelor și minimizarea disputei și a punctelor fierbinți
  2. Suprapunerea calculelor cu interacțiuni
  3. Replicarea datelor sau calculelor
  4. Utilizarea operațiunilor de interacțiune colectivă optimizate
  5. Suprapunerea interacțiunilor cu alte interacțiuni

Maximizarea localității datelor și minimizarea disputei și a punctelor fierbinți

- Tehnicile de îmbunătățire a localității de date cuprind o gamă largă de scheme care încearcă:
  - minimizarea volumul de date non-locale care sunt accesate,
  - maximizarea reutilizării datelor accesate recent și
  - minimizarea frecvenței acceselor.
- Minimizarea volumului schimbului de date:
  - prin utilizarea unor scheme de descompunere și mapare adecvate.
  - Exemplu: inmultirea matriceala
    - utilizând o mapare bidimensională a calculelor la procese, se reduce cantitatea de date partajate care trebuie accesate de fiecare sarcină, spre deosebire de o mapare unidimensională.
- Minimizarea frecvenței interacțiunilor:
  - prin restructurarea algoritmului astfel încât datele partajate să fie accesate și utilizate în bucăți mari.
- Minimizarea disputei și a punctelor fierbinți:
  - Disputa apare cand
    - mai multe sarcini încearcă să acceseze simultan aceleași resurse.
    - mai multe transmiteri simultane de date prin aceeași legătură de interconectare,
    - mai multe accesari simultane la același bloc de memorie,
    - mai multe procese care trimit mesaje către același proces în același timp

Suprapunerea calculelor cu interacțiuni

- Există o serie de tehnici care pot fi utilizate
- Cel mai simplu: inițiază o interacțiune suficient de devreme, astfel încât să fie finalizată înainte de a fi nevoie pentru calcul.
  - trebuie să se identifice calcule care pot fi efectuate înainte de interacțiune și nu depind de ea
  - programul paralel trebuie structurat pentru a iniția interacțiunea într-un moment anterior al execuției decât este necesar în algoritmul inițial.
- În anumite scheme de mapare dinamică, de îndată ce un proces rămâne fără lucru, acesta solicită și primește lucrări suplimentare de la un alt proces.
  - Dacă procesul poate anticipa că va rămâne fără muncă, va iniția în prealabil o interacțiune de transfer de muncă.
- In arhitectura spațiului de adresă partajat, asistat de preincarcarea bazata pe hardware.
  - Hardware-ul poate anticipa adresele de memorie la care va trebui accesat în viitorul imediat și poate iniția accesul înainte de momentul în care acestea vor fi necesare.

Replicarea datelor sau calculelor

- Replicarea datelor
  - Exemplu:
    - mai multe procese pot necesita accesul in citire frecventă la structura de date partajate, cum ar fi un tabel hash, într-un model neregulat.
      - reproducerea unui copii a structurii de date partajate pe fiecare proces
      - după interacțiunea inițială în timpul replicării, toate accesele ulterioare la această structură de date sunt libere de orice interacțiune.
  - Crește cerințele de memorie ale unui program paralel:
    - Cantitatea totală de memorie necesară pentru stocarea datelor replicate crește liniar cu nr. proceselor concurente.
    - Se poate limita dimensiunea problemei care poate fi rezolvată pe un computer paralel dat.
- Replicarea datelor
  - procesele dintr-un program paralel împărtășesc adesea rezultate intermediare.
    - În unele situații, poate fi mai rentabil pentru un proces să calculeze aceste rezultate intermediare decât să le obțină dintr-un alt proces care le generează.
  - Exemplu:
    - Transformata Fourier rapida, iN puteri distincte ale lui ω sunt calculate si utilizate in diverse portiuni ale calcului.
    - Într-o implementare paralelă a FFT, diferite procese necesită suprapuneri de subseturi ale acestor N factori.
    - Paradigma de transmitere a mesajelor: fiecare proces calculează local toți factorii de care are nevoie.
      - Deși algoritmul paralel poate efectua mult mai multe calcule ale factorilor decât algoritmul serial, acesta poate fi totuși mai rapid decât partajarea factorilor.

Utilizarea operaţiunilor de interacţiune colectivă optimizate

- Adesea, tiparele de interacțiune dintre activitățile concurente sunt statice și regulate.
- O clasă de astfel de tipare de interacțiune statică și regulată sunt cele care sunt efectuate de grupuri de sarcini și sunt utilizate pentru a obține accesuri obișnuite de date sau pentru a efectua anumite tipuri de calcule pe date distribuite.
- Au fost identificate o serie de astfel de operații de interacțiune colectivă care apar frecvent în mai mulți algoritmi paraleli.
- Exemple:
  - Difuzarea unor date la toate procesele sau
  - adunarea numerelor, fiecare aparținând unui proces diferit.
- Operațiunile colective de schimb de date pot fi clasificate în trei categorii.
  1. operațiunile utilizate de sarcinile pentru a accesa date,
  2. operațiunile sunt utilizate pentru a efectua unele calcule intensiv în comunicare,
  3. utilizat pentru sincronizare.
- Au fost dezvoltate implementări optimizate ale acestor operațiuni colective care reduc la minimum surplusul cauzat de transferul de date, precum și disputa.
- Implementările optimizate ale acestor operațiuni de interacțiune colectivă sunt disponibile sub formă de bibliotecă de la furnizorii majorității computerelor paralele, de exemplu, MPI.
  - designerul de algoritmi nu trebuie să se gândească la modul în care aceste operațiuni sunt puse în aplicare și trebuie să se concentreze numai pe funcționalitatea obținută de aceste operațiuni.

Suprapunerea interacțiunilor cu alte interacțiuni

- Suprapunerea interacțiunilor între mai multe perechi de procese poate reduce eficient volumul de comunicare.
- Exemplu:
  - operația de comunicare colectivă de difuzare unul-la-toti într-o paradigmă de transmitere a mesajelor cu patru procese P0, P1, P2 și P3.
  - Un algoritm utilizat frecvent pentru difuzarea unor date de la P0 la toate celelalte procese funcționează după cum urmează.
    - În primul pas, P0 trimite datele la P2.
    - În a doua etapă, P0 trimite datele la P1 și, simultan, P2 trimite aceleași date pe care le-a primit de la P0 la P3.
    - Întreaga operație este astfel completă în doi pași, deoarece cele două interacțiuni ale celui de-al doilea pas necesită o singură etapă.
    - Această operație este ilustrată în figura (a).
  - Un algoritm naivă de difuzarear trimite datele de la P0 la P1 la P2 la P3, consumând astfel trei pași așa cum este ilustrat în figura(b).