Algoritmi paraleli
Curs 4

Vlad Olaru

vlad.olaru@fmi.unibuc.ro

Calculatoare si Tehnologia Informatiei

Universitatea din Bucuresti

Metode iterative

$$f: R^n \to R^n$$
$$x(t+1) = f\big(x(t)\big) \quad t = 0,1, \ldots$$

- structura generala pt. sisteme de ecuatii, probleme de optimizare, etc
- se numesc *algoritmi iterativi* sau *metode de relaxare*
- uneori se foloseste notatia $x = f(x)$
- daca $\{x(t)\}$ converge la x^* si f continua, x^*se numeste *punct fix al lui f*

$$x^* = f(x^*)$$

- caz special, *algoritm iterativ liniar*:

$f(x) = A\,x + b$, unde A e matrice patrata si b vector

Paralelizarea metodelor iterative

$$f: R^n \to R^n$$
$$x_i(t+1) = f_i\big(x(t)\big) \quad t = 0,1,\dots$$

unde $x_i(t)$ si f_i sunt componentele i ale vectorilor x si f

- Q: este posibila paralelizarea acestui tipar de algoritmi?
- A: date fiind n procesoare, fiecare calculeaza cate un $x_i(t+1)$ si comunica rezultatul celorlalte procesoare pt iteratia urmatoare
- obs: in general, complexitatea in timp a acestor algoritmi nu e relevanta in absenta unui criteriu de terminare (valabil si pt. cazul secvential)

Metode (algoritmi) Jacobi

$$x_i(t+1) = f_i(x_1(t), \dots, x_n(t)) \quad i = 1, \dots, n$$

- toate componentele x_i ale lui $x(t)$ sunt actualizate *simultan*
- din nou, vom folosi o reprezentare de graf orientat aciclic (DAG), numit in acest caz *graf de dependente ("dependency graph"):*

  - set de noduri N = *{1, …n}* corespunzator componentelor lui x

  - arcul *(i,j)* exprima o dependenta ⇔ f_j depinde de x_i

  i.e., procesorul i trebuie sa comunice procesorului j valorile lui $x_i(t)$

Exemplu graf de dependente

$$x_1(t+1) = f_1(x_1(t), x_3(t))$$
$$x_2(t+1) = f_2(x_1(t), x_2(t))$$
$$x_3(t+1) = f_3(x_2(t), x_3(t), x_4(t))$$
$$x_4(t+1) = f_4(x_2(t), x_4(t))$$

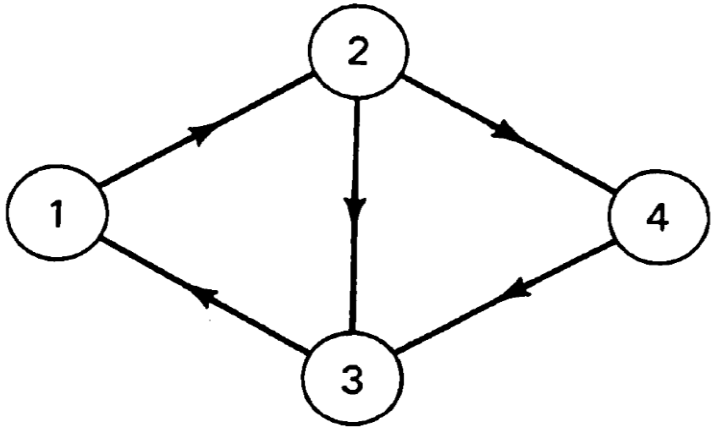


Figure 1.2.6 The dependency graph associated with an iteration of the form

$$x_1(t+1) = f_1\big(x_1(t), x_3(t)\big)$$
$$x_2(t+1) = f_2\big(x_1(t), x_2(t)\big)$$
$$x_3(t+1) = f_3\big(x_2(t), x_3(t), x_4(t)\big)$$
$$x_4(t+1) = f_4\big(x_2(t), x_4(t)\big).$$

Desfasurarea grafului de dependente in timp

- pp. iteratia se executa pt. $t = 0,1, \dots, T$ unde $T \in N$

$$x_1(t+1) = f_1(x_1(t), x_3(t))$$
$$x_2(t+1) = f_2(x_1(t), x_2(t))$$
$$x_3(t+1) = f_3(x_2(t), x_3(t), x_4(t))$$
$$x_4(t+1) = f_4(x_2(t), x_4(t))$$

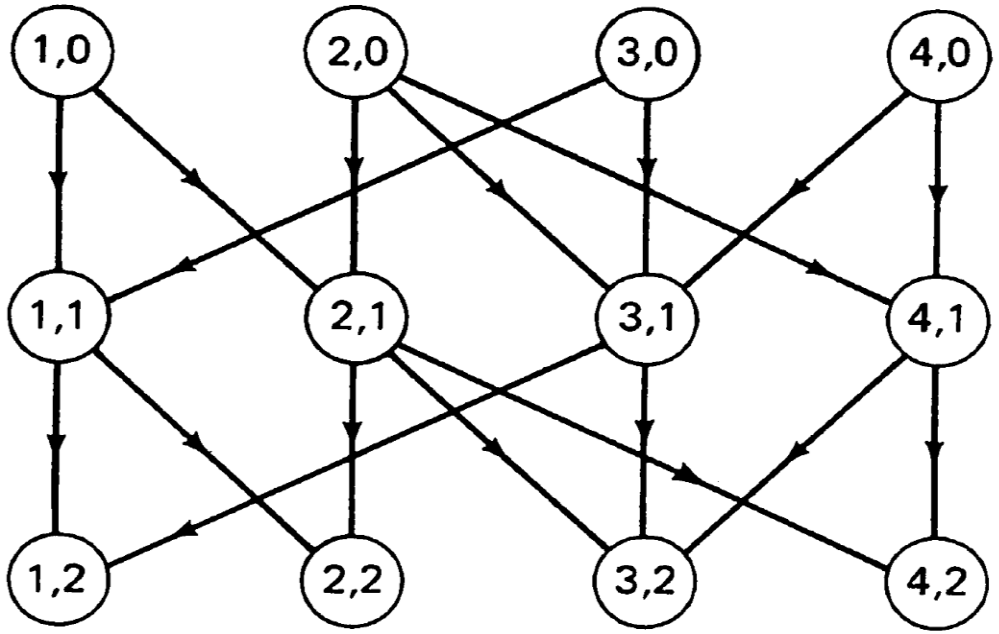


Figure 1.2.7 The DAG corresponding to two iterations when the function f has the dependency graph shown in Fig. 1.2.6. The nodes of the DAG are of the form (i, t), where $i \in \{1, \dots, n\}$, and t is the iteration count. The arcs are of the form $\big((i, t), (j, t+1)\big)$, where (i, j) is an arc of the dependency graph or $i = j$.

Metode iterative paralelizate cu componente bloc

- formulare de paralelism coarse-grain a iteratiei $x = f(x)$

$$x_j(t+1) = f_j\big(x(t)\big) \quad j = 1, \dots p$$

unde $x = (x_1, \dots, x_j, \dots, x_p)$ si x_j vector de dimensiune n_j (*componente bloc ale lui x*) a.i.

$$\sum_{j=1}^{p} n_j = \text{n}$$

$$f_j \colon R^n \to R^{n_j}$$

$$R^n = R^{n1} \; x \; \dots \; x \; R^{np}$$

(Obs: dependentele raman, nu se modifica decat granularitatea)

- fiecare componenta bloc x_j se asigneaza unuia dintre cele p procesoare care devine responsabil pt actualizarea componentei

=> algoritm *block-parallelized (bloc paralel)*

Metode iterative paralelizate cu componente bloc

- se redefineste graful de dependente $G=(N,A)$ cu
  - $N = \{1, \ldots, p\}$
  - $A = \{(i,j) \mid f_j \text{ depinde de } x_i\}$
- motivatii algoritm bloc-paralel:
  - nr de procesoare redus => plasarea mai multor x_i pe acelasi procesor
  - functiile scalare f_i pot avea operatii comune (optimizare)
  - reducerea comunicatiilor inter-procesoare (problema granularitatii)

Utilitate metode iterative

- paralelizarea unei iteratii prin asignarea actualizarilor unor procesoare diferite are sens cand actualizarea fiecarei componente x_i presupune un calcul diferit
- exemplu contrar

$$f_i(x_1(t), \dots, x_n(t)) = x_i + \sqrt{\sum_{j=1}^{n} {x_j}^2}$$

  - calculul sumei de catre fiecare procesor este nejustificat
  - poate fi facut de un singur procesor care sa comunice rezultatul celorlalte procesoare
  - alternativ, poate fi evaluat in paralel, colaborativ, folosind calcul prefix
- in general, calculul f_i implica putina duplicare (sau deloc) a calculului

Metode (algoritmi) Gauss-Seidel

$$x_i(t+1) = f_i(x_1(t+1), \ldots, x_{i-1}(t+1), x_i(t), \ldots, x_n(t))$$

- componentele x_i se actualizeaza pe rand, folosindu-se la fiecare iteratie cea mai noua valoare a celorlalte componente

=> *algoritm Gauss-Seidel bazat pe functia f*

- pt. ca incorporeaza cea mai noua informatie, pot converge mai repede decat metodele Jacobi !

- vom discuta posibilitatea paralelizarii unei singure iteratii Gauss-Seidel, asa numita operatie de *sweep*

- la limita, o asemenea iteratie poate fi complet ne-paralelizabila
  - ex: daca toate functiile f_i depind de toate componentele x_j, doar o componenta se poate actualiza la un moment dat

- daca insa graful de dependente e rar (*sparse*), exista posibilitatea executiei in paralel a mai multor actualizari

Exemplu iteratie Gauss-Seidel cu graf de dependenta sparse

$$x_i(t+1) = f_i(x_1(t+1), \ldots, x_{i-1}(t+1), x_i(t), \ldots, x_n(t))$$

- componentele x_3 si x_4 pot fi actualizate in paralel !
- sunt 4 actualizari, dar adancimea grafului este D = 3 !
- ordinea de actualizare influenteaza paralelismul

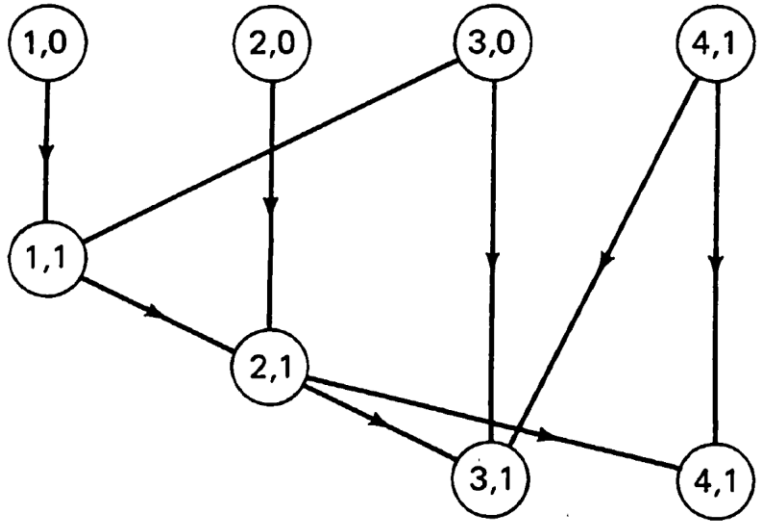


Figure 1.2.8 Illustration of the parallelization of Gauss–Seidel iterations. Let f be a function whose dependency graph is as in Fig. 1.2.6. The Gauss–Seidel algorithm based on f takes the form

$$\begin{aligned}
x_1(t+1) &= f_1\big(x_1(t), x_3(t)\big)\\
x_2(t+1) &= f_2\big(x_1(t+1), x_2(t)\big)\\
x_3(t+1) &= f_3\big(x_2(t+1), x_3(t), x_4(t)\big)\\
x_4(t+1) &= f_4\big(x_2(t+1), x_4(t)\big).
\end{aligned}$$

Ordinea actualizarilor in algoritmii Gauss-Seidel

- pt aceeasi functie *f* exista mai multi algoritmi GS data fiind libertatea de a schimba ordinea actualizarilor
  - ex: executam actualizarile de la coada la capat (de la x_n la x_1)
- ordini diferite ale actualizarilor rezulta in general in algortimi GS diferiti, cu rezultate diferite
- exista si aplicatii insa in care un algoritm GS converge in limita unui nr mare de iteratii la aceeasi valoare, indiferent de ordinea actualizarilor
  - daca viteza de convergenta aferenta diferitelor ordini de actualizare nu difera mult => *vom alege ordinea de actualizare care permite maximum de paralelism* !

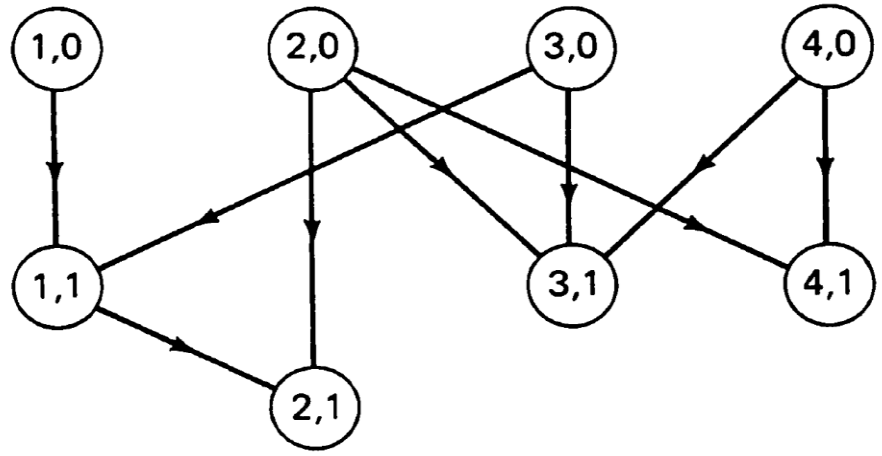
Cresterea paralelismului prin schimbarea ordinii actualizarilor GS

$$x_i(t+1) = f_i(x_1(t+1), \dots, x_{i-1}(t+1), x_i(t), \dots, x_n(t))$$

- daca schimbam ordinea de actualizare si consideram:

$$x_1(t+1) = f_1(x_1(t), x_3(t))$$
$$x_3(t+1) = f_3(x_2(t), x_3(t), x_4(t))$$
$$x_4(t+1) = f_4(x_2(t), x_4(t))$$
$$x_2(t+1) = f_2(x_1(t+1), x_2(t))$$

- adancimea grafului D devine 2
- se observa ca un *sweep* se poate face in paralel in doi pasi cu doua procesoare



Determinarea ordinii actualizarilor pt. paralelizarea optima a operatiei de sweep

- graf de dependenta $G = (N,A)$
- colorarea G cu K culori (fiecare nod din G are asociata o culoare) din multimea $\{1, \ldots K\}$

$$h(i) = k \qquad \forall\, i \in N$$

- idee: variabilele colorate cu aceeasi culoare se pot actualiza in paralel
- in aceste conditii, maximizarea paralelismului revine la gasirea unei colorari optimale a grafului G
- **Propozitie**

  (i) exista o ordine de actualizare a variabilelor intr-o iteratie Gauss-Seidel care se poate efectua in K pasi paraleli

  $\Leftrightarrow$

  (*ii)* exista o colorare a grafului de dependenta G cu K culori a.i. nu exista cicluri cu toate nodurile de aceeasi culoare

Demonstratie *(i)* => *(ii)*

- pp. o ordine de actualizare a variabilelor unei iteratii care necesita K pasi paraleli
- fie culoarea nodului i, $h(i) = k$, daca x_i se actualizeaza in pasul paralel k
- fie un ciclu pozitiv $i_1, i_2, \ldots i_m$, cu $i_1 = i_m$
- fie i_l un nod in ciclu $(1 \leq i_l < m)$ care apare primul in ordinea actualizarilor
- i_{l+1} se actualizeaza dupa i_l *si* $(i_l, i_{l+1}) \in A$ => $x_{il+1}(t+1)$ *depinde de* $x_{il}(t+1)$

$\Rightarrow$ cele doua noduri nu se pot actualiza simultan = > $h(i_l) \neq h(i_{l+1})$

$\Rightarrow$ in orice ciclu pozitiv exista doua noduri colorate diferit, Qed

Lema

- $G = (N, A)$ DAG, daca $(i,j) \in A$ exista o ordine de actualizare a nodurilor a.i. j sa apara inainte de i
- demonstratie
  - $\forall\, i$ nod, fie d_i = max nr de arce dintr-o cale pozitiva care porneste de la i
  - G aciclic => d_i finit
  - ordonam nodurile in ordinea crescatoare a d_i
  - $(i,j) \in A$ => $d_i > d_j$ => j se actualizeaza inaintea lui i

Demonstratie *(ii)* => *(i)*

- pp. o colorare h a lui G cu K culori si niciun ciclu pozitiv cu toate nodurile de aceeasi culoare
- pt. fiecare culoare k, def G_k, subgraf al lui G cu noduri de culoare k si arcele dintre ele
- G_k aciclic prin ipoteza => cf lemei anterioare nodurile din G_k se pot ordona a.i. j se actualizeaza inainte de i daca $(i,j) \in A$
- se ordoneaza G in ordinea crescatoare a culorilor
- fie o iteratie GS cf acestei ordonari si doua noduri i si j, $i \neq j$, a.i. $h(i) = h(j) = k$
- cazuri posibile
  - (i,j) si (j,i) nu sunt arce => x_i si x_j se pot actualiza in paralel
  - $(i,j) \in A$ si $(j,i) \in A$ imposibil pt. ca G_k aciclic
  - $(i,j) \in A$ si (j,i) nu e arc => j apare inainte de i in ordinea actualizarilor => $x_j(t+1)$ necesita doar valoarea lui $x_i(t)$ nu si $x_i(t+1)$; similar pt cazul simetric

  => $\forall\, x_i$ cu aceeasi culoare se pot actualiza in paralel

Exemplu

- doua culori sunt suficiente, eg *h(1)=h(3)=h(4)=1* si *h(2)=2*
- toate ciclurile pozitive trec prin 2 => ∄ ciclu pozitiv cu noduri de aceeasi culoare
- G_1 aciclic, $d_1 = 0$, $d_3 = 1$, $d_4 = 2$ => ordonarea actualizarilor cf demonstratiei este (1, 3, 4, 2)

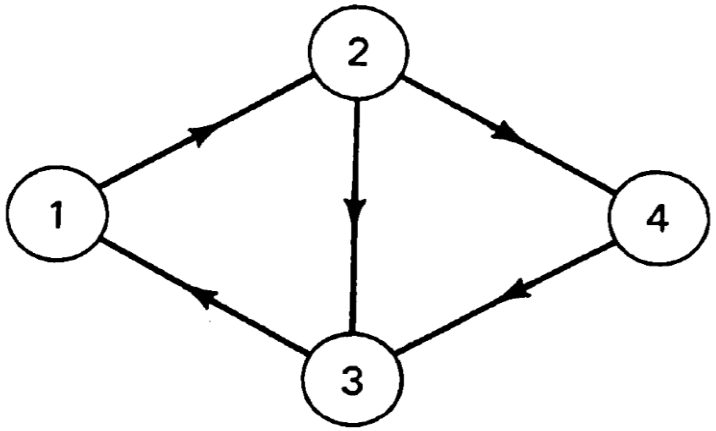


Figure 1.2.6 The dependency graph associated with an iteration of the form

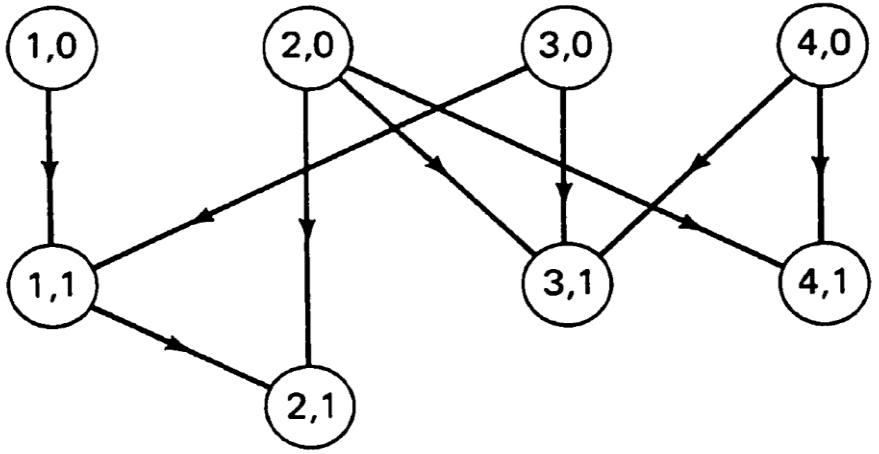
$$x_1(t+1) = f_1\left(x_1(t), x_3(t)\right)$$
$$x_2(t+1) = f_2\left(x_1(t), x_2(t)\right)$$
$$x_3(t+1) = f_3\left(x_2(t), x_3(t), x_4(t)\right)$$
$$x_4(t+1) = f_4\left(x_2(t), x_4(t)\right).$$

Exemplu iteratie GS optima

- ordonarea actualizarilor cf demonstratiei este (1, 3, 4, 2)

=> iteratia Gauss-Seidel corespunzatoare acestei ordini este cea de mai jos

(se poate verifica vizual ca ordinea aceasta necesita nr min de pasi paraleli per sweep)



Caz simplificat de paralelizare optima a unei iteratii GS

- pp. $(i,j) \in A \Leftrightarrow (j,i) \in A$ (proprietate de simetrie). Atunci,

(i) exista o ordine de actualizare a unei iteratii GS care poate fi facuta in K pasi paraleli

$\Leftrightarrow$

(ii) exista o colorare a grafului de dependente cu cel mult K culori a.i. nodurile adiacente au culori diferite (i.e., $(i,j) \in A \Rightarrow h(i) \neq h(j)$)

- demonstratie
  - suficient sa demonstram ca *(ii)* e echivalent cu *(ii)* din propozitia anterioara

  "=>" pp $\exists$ ciclu pozitiv cu toate nodurile de aceeasi culoare => $\exists$ doua noduri adiacente de aceeasi culoare

  "<=" $(i,j) \in A$ si $(j,i) \in A$ => (i,j) si (j,i) formeaza un ciclu pozitiv => daca doua noduri adiacente au aceeasi culoare $\exists$ ciclu pozitiv cu toate nodurile de aceeasi culoare

Consecinte

- colorarea grafurilor cu $K > 3$ culori e NP-complete => gasirea unei ordini a actualizarilor variabilelor care sa maximizeze paralelismul unei iteratii Gauss-Seidel e o problema dificila
- in practica, exista probleme cu structura speciala in care o colorare cu cateva culori poate fi gasita usor
- (1) graf *G* neorientat, daca *G* e arbore sunt suficiente doua culori
  - alegem un nod arbitrar din arbore
  - asociem culoarea 1 (respectiv 2) tuturor nodurilor din G care pot fi vizitate incepand din nodul respectiv traversand un nr par (respective impar) de arce
- (2) daca orice nod din *G* are cel mult *D* vecini, *D+1* culori sunt suficiente
  - coloram nodurile pe rand
  - pp. primele *i* noduri colorate si dorim sa coloram *i+1*
  - folosim D+1 culori => ∃ o culoare pe care o putem folosi pt *i+1* a.i. e colorat diferit de vecinii deja colorati ai lui *i+1*

Paralelizarea metodelor iterative

Reamintim: *Scopul paralelizarii algoritmilor iterativi este de a obtine reducere cat mai semnificativa a complexitatii in timp*

Pentru o accelerare optima avem nevoie de:

- planificator eficient
- timp de comunicatie redus

Pana acum nu am luat in calcul comunicatia in sistemele paralele!

Analiza comunicatiei

- comunicatia inter-procesor reprezinta o fractiune importanta din timpul de rulare al programelor paralele
- interpretare posibila: comunicatia are un cost de penalizare (echivalent, exista intarzieri de comunicare)

$$CP = \frac{T_{total}}{T_{comp}}$$

T_{total} este timpul total de executie al programului

T_{comp} este timpul alocat exclusiv calculului, fara comunicare (i.e., daca consideram comunicarea instantanee)

Cadrul analizei

- vom considera o retea de procesoare interconectata cu legaturi de comunicatie (MIMD fara memorie partajata)
- fiecare procesor are memorie locala si schimba informatii cu alte procesoare prin retea
- informatiile sunt structurate ca pachete de biti de lungimi diferite
- vom considera de asemenea paradigma *store-and-forward* pentru comunicarea pachetelor
- pp. ca bitii unui pachet sunt transmisi consecutiv fara intrerupere
- pp. un model de comunicare direct procesor-la-procesor

Surse de intarziere in comunicare

- timp procesare comunicatie:
  - timpul necesar pregatirii informatiei pt transmisie
  - e.g. asamblare/dezasamblare informatii in pachete, adaugarea header-elor, selectia rutei, mutarea pachetelor in bufferele de transmisie, etc
- timp de asteptare in coada:
  - pachetele asamblate pt transmisie sunt puse intr-o coada de asteptare inainte de inceperea transmisiei
  - motive:
    - planificarea accesului pachetelor la link in conditii de competitie la resurse
    - amanarea transmisiei pana cand destinatia poate primi pachetul (eg, sliding window protocols)
    - retransmisii in caz de eroare
- timp de transmisie: durata transmisiei tuturor bitilor unui pachet
- timp de propagare: durata de transfer a ultimului bit al pachetului

Calcul timp de comunicare

- oricare dintre timpii anterior pot fi neglijabili
  - informatie generata cu regularitate si resurse suficiente => timpul de asteptare in coada e neglijabil
  - distanta fizica intre sursa si destinatie este f. mica => timpul de propagare e neglijabil
- simplificari rezonabile
  - timp de procesare si propagare constant (P)
  - timp de transmisie proportional cu nr de biti (R pentru 1 bit)
- intarzierea unui pachet pe o linie de comunicatie

$$D = P + RL + Q$$

- L = lungimea in biti a pachetului
- Q = timp de asteptare in coada, in general dificil de modelat

Observatii

$$D = P + RL + Q$$

- in majoritatea retelelor moderne, $P + RL$ este semnificativ mai mare decat executia unei operatii numerice elementare (e.g. inmultire in virgula-mobila)
  - daca un algoritm foloseste comunicatia intensiv, timpul de comunicare domina timpul de executie (complexitatea timp) => performanta redusa (echivalent, costul de penalizare CP va fi mare)
- se poate incerca **paralelizarea comunicatiei**

Mesaje

- pachetele sunt unitatea de comunicare in modelul nostru
- in general ele fac parte dintr-un mesaj care are sens doar in intregimea sa, deci intarzierea intregului mesaj devine importanta
- aceasta intarziere depinde de felul in care este impartit mesajul in pachete si modul in care se trimit pachetele individuale
- ex: un mesaj se divide in *n* pachete de lungimi egale
  - presupunem timpul de transmisie al unui pachet T
  - transmisia pachetelor in paralel pe *n* cai separate care au acelasi timp de transmisie per pachet rezulta intr-o reducere de *n* ori a timpului de comunicatie comparativ cu transmisia pe o singura cale
    - pp overhead-ul segmentarii in pachete, procesarea, propagarea si intarzierea in cozi neglijabile
    - ex. SCTP vs TCP (la nivel de protocol)

Transmisia mesajelor in paralel

- alta posibilitate de paralelizare a comunicatiei: *pipelining*
- mesaj impartit in n pachete de lungimi egale
- pachetele se transmit secvential pe o cale de lungime $k > 1$
- timp de transmisie de la nod la nod T (per pachet)
- timp total de transmisie a mesajului in mod pipeline ca o colectie de n pachete:

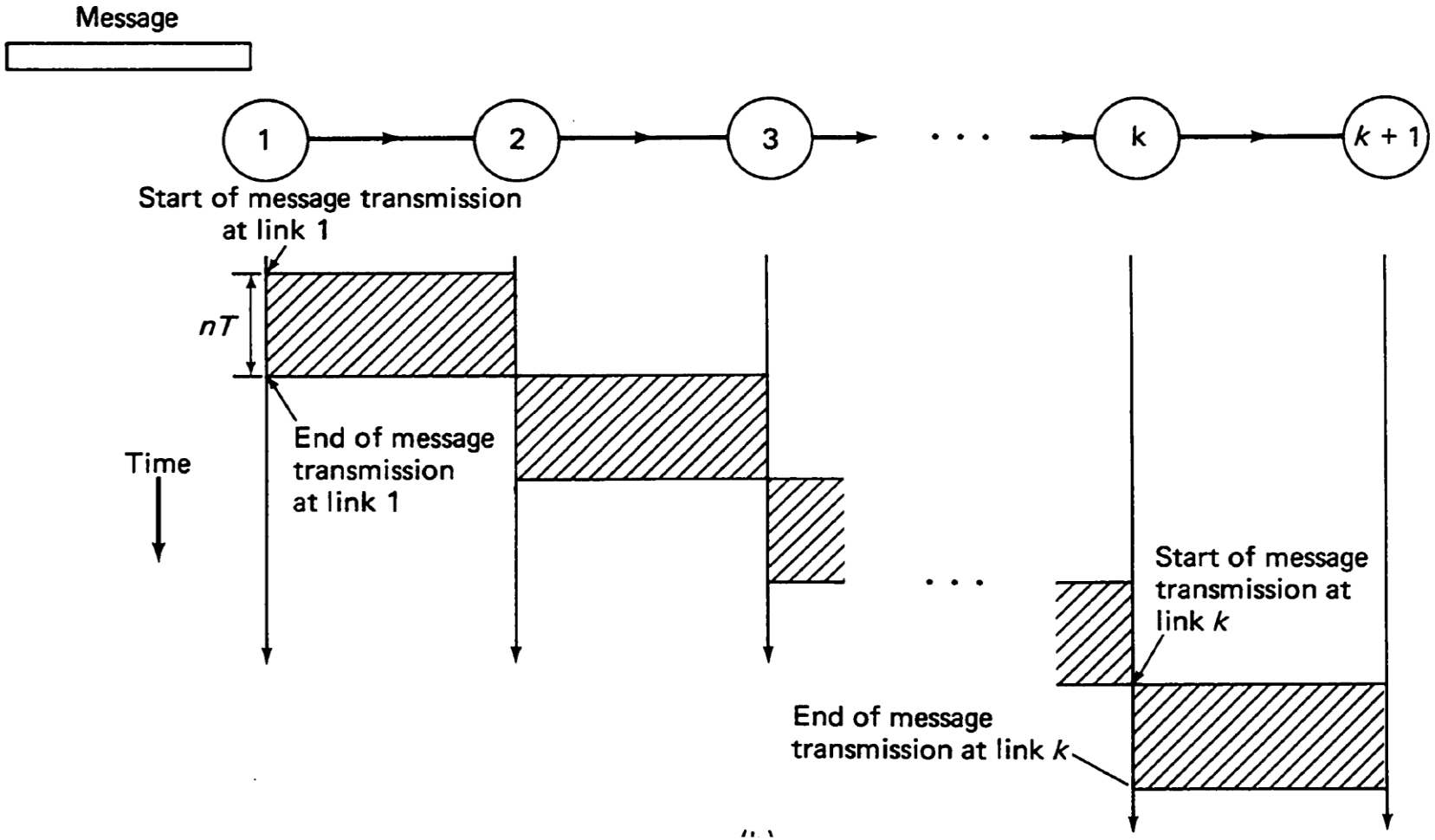
$$(n - k + 1)T$$

- timp total de transmisie mesaj ca un singur pachet de dimensiune n:

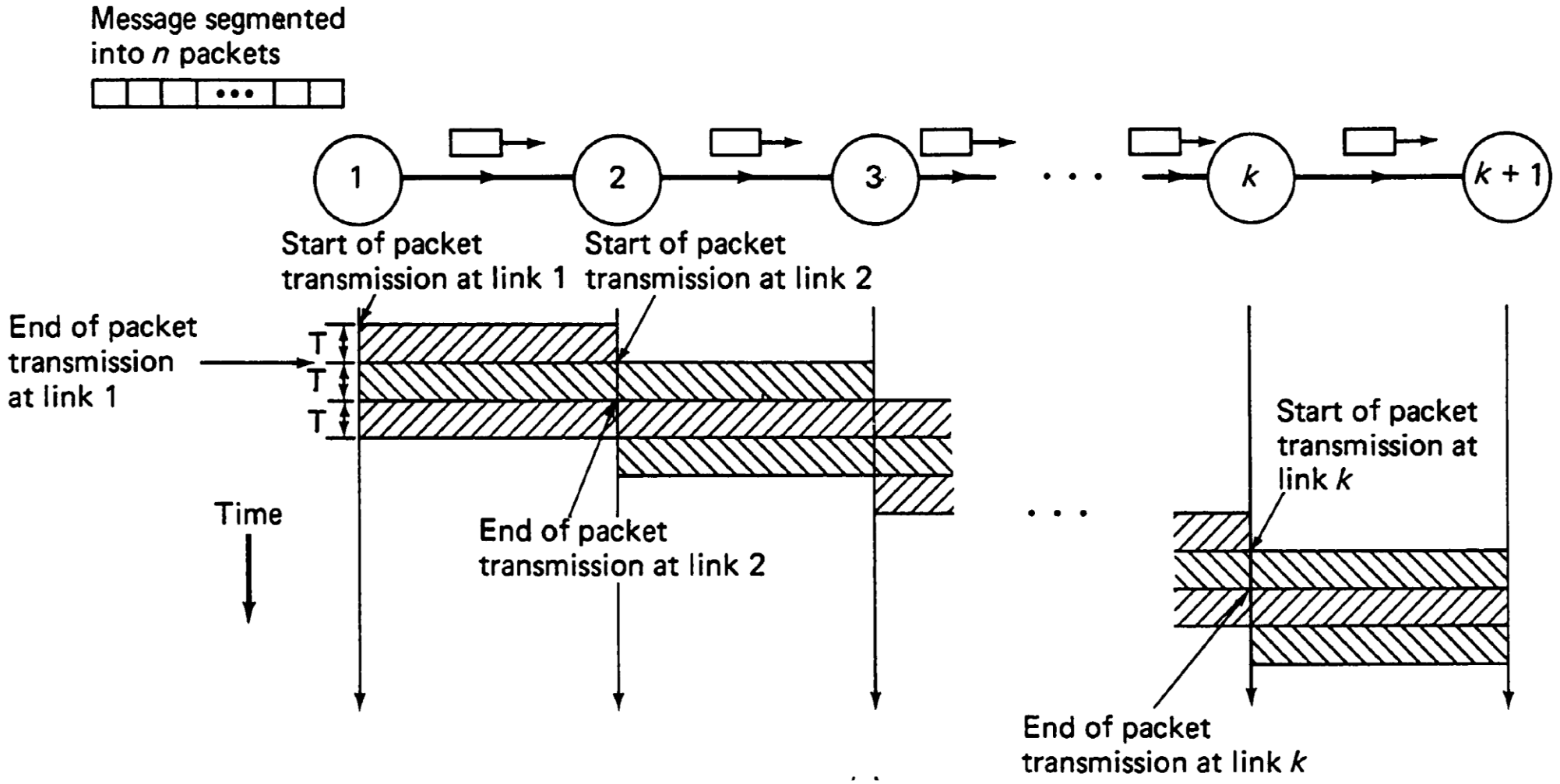
$$nkT$$

- Obs: pp neglijabile efectele overhead-ului, procesarii, propagarii si asteptarii in cozi
  - de asemenea, pp un nod primeste un pachet in intregime inainte sa trimita mai departe orice parte a lui

Transmisie mesaj ca un singur pachet



Pachete trimise in mod pipeline



Observatii

- daca pachetele sunt f. mici => intarzierea se poate reduce cu un factor egal cu nr de link-uri (linii de comunicatie) ≈ timpul de transmisie a mesajului pe un singur link

⇒ alternativa, *transmisia cut-through*
  - un nod poate transmite altui nod o portiune a pachetului fara sa astepte primirea in intregime a pachetului
  - revine la segmentarea pachetului in multe pachete mici pt a beneficia de avantajele pipelining-ului

- pipelining-ul e aplicabil si in alte situatii, de ex peste un spanning tree (arbore de acoperire)

Factori care influenteaza intarzierile de comunicatie

- algoritmii care controleaza reteaua de comunicatie
  - controlul erorilor
  - rutarea
  - flow control
- topologia retelei de comunicatie
  - numarul, natura si locul (pozitia) liniilor de comunicare
- structura problemei si proiectarea algoritmului a.i. sa se potriveasca acestei structuri (inclusiv gradul de sincronizare necesar algoritmului)