Algoritmi paraleli
Curs 3

Vlad Olaru

vlad.olaru@fmi.unibuc.ro

Calculatoare si Tehnologia Informatiei

Universitatea din Bucuresti

Modele formale

- varietate de modele de calcul paralel fiecare cu propriile presupuneri despre
  - puterea de calcul a procesoarelor
  - mecanismul de comunicare interprocesor
- ex presupuneri:
  - fiecare procesor poate executa instructiuni de baza (operatii aritmetice, comparatii, if-then-else)
  - fiecare instructiune ia o unitate de timp
  - exista un mecanism de comunicare interprocesor
  - cel mai adesea, operatiile de transfer de date sunt instantanee si fara costuri asociate
  - uneori pp ca folosim MIMD cu memorie partajata, alteori MIMD message-passing
    - pt schimbul de mesaje vom face pp asupra intarzierilor mesajelor in retea
- in continuare vom folosi un model simplu pt. ilustrarea unor aspecte importante ale calculului paralel
  - adecvat pt majoritatea algoritmilor sincroni despre care vom vorbi, cata vreme se ignora comunicatia interprocesor

Reprezentarea algoritmilor paraleli folosind grafuri aciclice orientate (DAG)

- grafuri aciclice orientate (DAG, directed acyclic graph) = graf orientat fara cicluri pozitive (formate din arce cu acelasi sens)

G(V,E)

- V este multimea nodurilor, care denota **operatiile** efectuate de algoritm
- E este multimea muchiilor, ce reprezinta **dependenta dintre date**
- $(i, j) \in E$: operatia nodului j foloseste rezultatele operatiei nodului i (nodul i este *predecesorul* lui j)
- operatia poate fi elementara (operatie aritmetica, booleana, etc) sau operatie de nivel inalt (eg, subrutina)

Terminologie DAG

- *in-degree (i)* = nr de predecesori ai nodului *i*
- *out-degree(i)* = nr de noduri pt care *i* este predecesor
- nodurile cu *in-degree* zero = noduri de intrare (*input*)
- noduri cu *out-degree* zero = noduri de iesire (*output*)
- cale pozitiva: secventa $i_0, \dots, i_K$ $a.i.$ $(i_k, ik_{+1}) \in E$ pt $k = 0..K-1$
  - lungimea caii = K
- adancimea D a unui DAG:
  - calea pozitiva de lungime maxima din graf
  - leaga un nod input de un nod output
  - valoare finita (consecinta a lipsei ciclurilor din graf)
- obs: pp. D ≥ 1 (adica exista cel putin o muchie/un arc)

Exemplu

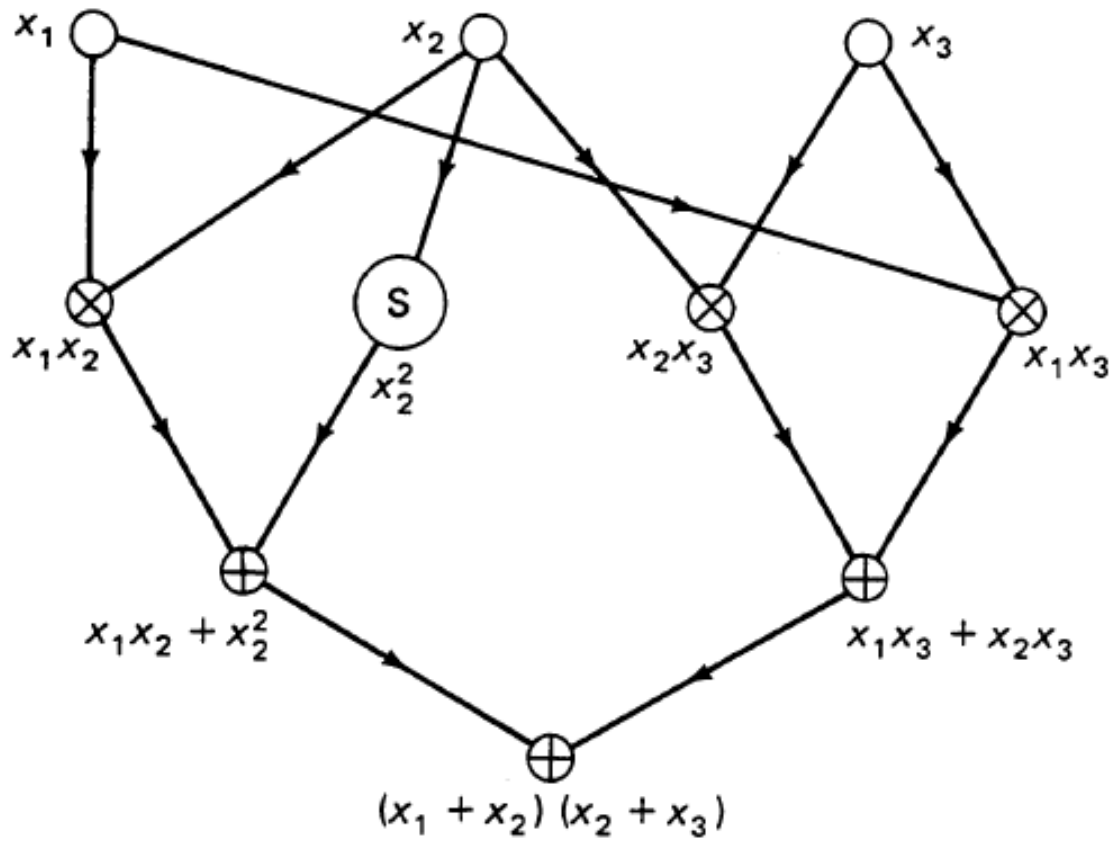


Figure 1.2.1 Representation of an algorithm for evaluating the arithmetic expression $(x_1 + x_2)(x_2 + x_3)$ by means of a DAG. The label at each node indicates the operation corresponding to that node. In particular, the label S stands for squaring.

Ipoteze

- DAG = reprezentare partiala a unui algoritm paralel
  - specifica operatiile care se executa, operanzii si constrangeri de precedenta in executia operatiilor
- 1) operatiile dureaza o singura unitate de timp (op. elementare, i.e. op. aritmetice)

$$x_i = f_i(\{\, x_j \mid j \in in\text{-}degree\ (i)\})$$

  unde f_i = operatia corespunzatoare nodului i (NB: nodul i este nod intern sau de output, nodurile de input sunt variabile externe de intrare)

- 2) pp. operatia asociata unui nod de intrare este citirea valorii, neglijabila ca durata, iar evaluarea f_i pt noduri interne si de output dureaza o unitate de timp
  - NB: in general, operatiile nodurilor pot dura mai mult de o unitate de timp
- 3) exista p procesoare disponibile si fiecare poate executa operatiile dorite

Ipoteze (cont.)

- 4) operatiile nodului i (nod intern sau de output) sunt efectuate de procesorul P_i si terminate la $t_i > 0$
- nodurilor interne i nu li se asociaza procesoare si $t_i = 0$
- 5) cel mult o operatie la un moment dat, i.e.

$$\text{i} \neq \text{j} \;\&\; t_i = t_j \Rightarrow P_i \neq P_j$$

unde i, j sunt noduri interne sau de output

- 6) $(i, j) \in E \Rightarrow\ t_j \geq t_i + 1$
  - operatia nodului j nu poate incepe decat dupa ce se termina operatia nodului i

Planificare

- dupa stabilirea P_i si t_i, se spune ca DAG-ul a fost planificat pentru executie paralela
- multimea $\{(i, P_i, t_i) | i \; nod \; intern \; sau \; de \; output\}$ se numeste *schedule* (*program*, *planificare*)
- intepretare a posibilelor implementari
  - memorie partajata (shared memory) MPs
    - nodurile (procesoarele) care executa operatiile pot stoca rezultate in memoria partajata in folosul celorlalte procesoare
  - sisteme message-passing
    - un arc *(i,j)* reprezinta in fapt transmisia unui mesaj
  - modelul folosit nu include (neglijeaza) costul accesului la memorie, in cazul general trebuie considerat si el
    - daca $P_i = P_j$ atunci $t_j \geq t_i + 1$
    - daca $P_i \neq P_j$ atunci $t_j \geq t_i + 1 + t_{acces\_mem}$
  - obs: t_{acces_mem} depinde de pozitia procesoarelor P_i si P_j in reteaua de interconectare

Masuri de complexitate

- numarul de procesoare
- timpul de executie (*complexitate in timp*)
- numarul de mesaje transmise in timpul executiei (*complexitate de comunicatie*)

E.g.: $O(\log(n))$, $O(n^2)$, $O(n\log(n))$

Reamintim:

- $f(x) = O(g(x)) \Leftrightarrow \exists c, x_0, \;\; a.i. \; f(x) \leq cg(x), \forall x \geq x_0$
- $f(x) = \Omega(g(x)) \Leftrightarrow \exists c, x_0, \;\; a.i. \; f(x) \geq cg(x), \forall x \geq x_0$
- $f(x) = \Theta(g(x)) \Leftrightarrow f(x) = O(g(x)) \;\&\&\; f(x) = \Omega(g(x))$

Masuri de complexitate (cont.)

- uzual exprimate ca functii dependente de *dimensiunea* problemei
  - informal, numarul de date de intrare
  - ex: n pt. adunarea a n date scalare
- Obs: chiar daca dimensiunea problemei e constanta, e posibil ca resursele folosite sa depinda de valoarea datelor de intrare

  => pt o problema de dimensiune data, trebuie calculate resursele necesare in cel mai defavorabil caz pt. toate combinatiile posibile ale datelor de intrare
- aspecte subtile
  - e posibil ca toate procesoarele sa fi terminat calcului, fara sa o stie insa
  - in acest caz, complexitatea include si timpul necesar procesoarelor sa decida terminarea algoritmului

Complexitate in timp (DAG)

- fie **G(V,E)** un DAG ce reprezinta un algoritm paralel
- fie $S = \{(i, P_i, t_i) | \ i \ nod \ intern \ sau \ de \ output\}$ o *planificare (schedule)* a DAG-ului pentru p procesoare
- timpul aferent executiei planificarii este $\mathbf{\max_i t_i}$
- **complexitatea in timp T_p** = timpul minim cheltuit pentru executia celei mai bune planificari care foloseste p procesoare

$$T_p = \min_S \max_i t_i$$

Timpul de executie minim cand avem p procesoare la dispozitie, complexitatea in timp a algoritmului descris de G

Complexitate de timp (DAG)

- T_1 = complexitatea in timp seriala (1 procesor), i.e. nr de noduri din G care nu sunt noduri input
- T_∞ = complexitatea in timp a algoritmului specificat de G, dat fiind un numar suficient de mare de procesoare (cel putin p^*)

$$T_\infty = \min_{p \geq 1} T_p$$

$$\text{i.e., } \exists\, p^*\ a.i.\ T_p = T_\infty\ \forall\, p \geq p^*$$

Obs: T_p = functie necrescatoare de p, limitata inferior de 0

- in general, $T_\infty \leq T_p \leq T_1$
- T_∞ = adancimea ("depth"-ul) G (T_∞ = D, cf. definitiei anterioare)

Demonstratie $T_\infty = \mathrm{D}$

=>

- fie $i_0, \ldots, i_k$ cea mai lunga cale (pozitiva) din G => i_0 nod de intrare si $k = D$ (cf. definitiei lui D)
- $\forall\, S$ planificare, $t_{i0} = 0$ si $t_{j+1} \geq t_j + 1\ \forall\, j = i_0, \ldots, i_{k-1}$

  => $t_{ik} \geq k = D$ => $T_\infty \geq D$

<=

- fie procesorul P_i asignat nodului i
- t_i = nr. arce al celei mai lungi cai (pozitive) de la un nod de intrare la i ($t_i = 0$ pt i nod de intrare)
- $(i, j) \in E$ => $t_j \geq t_i + 1$ (adaug un arc la cea mai lunga cale catre i)

  => planificare S valida si timpul corespunzator S este

  $$\max_i t_i = \mathrm{D}$$

  => $T_\infty \leq D$

Proprietati T_p

- pt. nr arbitrar de procesoare p

$$T_1 \geq T_p \geq T_\infty$$

- in general, T_p e greu de calculat
  - pp. un DAG particular si o anumita valoare a lui p
  - problema complicata de combinatorica
- totusi, acest aspect nu constitutie o problema, exista limite usor de folosit

Complexitate in timp DAG – Limite

Proposition 2.1. Suppose that for some output node i, there exists a positive path from every input node to i. Furthermore, suppose that the in–degree of each node is at most 2. Then,

$$T_\infty \geq \log n,$$

where n is the number of input nodes.

- **oricate procesoare avem la dispozitie, nu putem scadea sub complexitatea logaritmica!**
- analogie sorting networks: sortarea paralela are lower bound O($log(n)$) (Ajtai, Komlos, Szemeredi 1983) vs. sortare secventiala O($nlog(n)$)

Demonstratie

- inductie dupa nr de noduri input k pt care exista o cale catre un nod j (pt un nod input pp. ca exista o cale catre el insusi)
- demonstram ca $t_j \geq \log k$ pt. orice nod j pt. care exista o cale de la k noduri de input si pt orice schedule (planificare)
- $k = 1$, $t_j \geq 0$ adevarat pt. orice nod j
- pas inductiv:
  - pp ipoteza inductiva adevarata pt orice $k \leq m$ si consideram un nod j catre care exista cai dinspre $m+1$ noduri input
  - pt ca j nu poate avea mai mult de 2 predecesori, unul dintre ei, i, are cai de la $ceiling((m+1)/2)$ noduri input, deci

$$t_j \geq t_i + 1 \geq log(ceiling((m+1)/2)) + 1 \geq log\ (m+1)$$

Complexitate in timp - Limite

- **Q:** ce se intampla daca nr de procesoare e redus cu un anumit factor?
  A: timpul de executie creste cu cel mult acel factor

- **Prop 2.2** Pt $c \in N$ $(c \neq 0)$ si $q = cp$ => $T_p \leq cT_q$
- demonstratie:
  - fie planificarea S care necesita T_q timp pentru q procesoare
  - la fiecare pas (faza) se executa cel mult q operatii
  - pt p procesoare, faza se poate executa in max $q/p = c$ unitati de timp

  => planificare cu p procesoare care se executa in cel mult cT_q

Complexitate in timp - Limite

- **Prop 2.3** Pt. orice nr de procesoare p, $T_p < T_\infty + T_1 / p$
- demonstratie:
  - fie S schedule pt T_∞
  - fie n_t numarul de noduri i pt. care $t_i = t \in$ N (i.e., nr de operatii terminate la momentul t)
  - fie un schedule S_p pentru p procesoare care in faza t executa operatiile planificate la timpul t in S
  - faza t din S_p dureaza $ceiling(n_t / p)$ unitati de timp

  $\Rightarrow T_p$ este mai mic decat suma acestor unitati de timp pentru fiecare din pasii planificarii S_p

$$T_p \leq \sum_{t=1}^{T_\infty} ceiling(\frac{n_t}{p})$$

  - dar,

    $ceiling(n_t / p) < n_t / p + 1$ si

    $T_1 = sum(n_t)$ = suma tuturor operatiilor din nodurile interne si de output din G

$$\Rightarrow \quad T_p \leq \sum_{t=1}^{T_\infty} ceiling(\frac{n_t}{p}) \leq \sum_{t=1}^{T_\infty} (\frac{n_t}{p} + 1) = \frac{T_1}{p} + T_\infty$$

Corolar important

Proposition 2.4. (a) If $p \geq T_1/T_\infty$, then $T_p < 2T_\infty$. More generally, if $p = \Omega(T_1/T_\infty)$, then $T_p = O(T_\infty)$.
(b) If $p \leq T_1/T_\infty$, then

$$\frac{T_1}{p} \leq T_p < 2\frac{T_1}{p}.$$

More generally, if $p = O(T_1/T_\infty)$, then $T_p = \Theta(T_1/p)$.

- demonstratie (a)

  $p \geq T_1 / T_\infty \Rightarrow T_1 / p \leq T_\infty$

  Cf. 2.3 $\Rightarrow T_p < T_\infty + T_1 / p$

  $=> T_p < T_\infty + T_\infty = 2\, T_\infty$

Corolar important

Proposition 2.4. (a) If $p \geq T_1/T_\infty$, then $T_p < 2T_\infty$. More generally, if $p = \Omega(T_1/T_\infty)$, then $T_p = O(T_\infty)$.
(b) If $p \leq T_1/T_\infty$, then

$$\frac{T_1}{p} \leq T_p < 2\frac{T_1}{p}.$$

More generally, if $p = O(T_1/T_\infty)$, then $T_p = \Theta(T_1/p)$.

- demonstratie (b)

  p ≤ T_1 / T_∞ ⇒ $T_\infty \leq T_1$ / p

  Cf. 2.3 ⇒ $T_p < T_\infty + T_1$ / p

  ⇒ $T_p < 2\,T_1 / p$

  In plus, cf .2.2 (pt $q/p = c$ avem $T_p \leq cT_q$) si $c = p \Rightarrow T_1 \leq p\,T_p$

  ⇒ $T_1 / p \leq T_p$

Limitari fundamentale si concluzii

- 2.4 (a) => desi defininia T_∞ pp. un numar infinit de procesoare, $p = \Omega(T_1/T_\infty)$ procesoare sunt suficiente pentru atinge T_∞ in limita unui factor constant

$$T_\infty < T_p < 2\ T_\infty$$

- o planificare corespunzatoare se poate face prin simpla modificare a planificarii optimale pt. un numar nelimitat de procesoare, fara a rezolva o problema complicata de planificare
  - metodologie:
    - se dezvolta un algoritm paralel ca si cand sunt disponibile o infinitate de procesoare
    - apoi se adapteaza algoritmul pentru numarul de procesoare existente

- 2.4 (b) => daca $p = O(T_1/T_\infty)$ procesoare sunt disponibile atunci se poate reduce timpul de rulare cu un factor proportional cu p, aproape de optimalitate in limita unui factor constant

$$T_1 \text{ / p} \leq\ T_p \leq 2\ T_1 \text{ / p}$$

- **Concluzie**: cu un nr de procesoare p aproape egal cu T_1/T_∞ se pot obtine atat timp de executie optimal cat si accelerare (speed-up) optimala

Alegerea unui DAG optimal

- aceeasi problema computationala poate fi calculata diferit (DAG-uri diferite)
- cautam DAG-ul care minimizeaza T_p
- fie T_p^* valoarea lui T_p pentru DAG-ul optimal => T_p^* e o masura a complexitatii problemei, in timp ce T_p e complexitatea unui algoritm particular
- in general, evaluarea T_p^* e foarte dificila

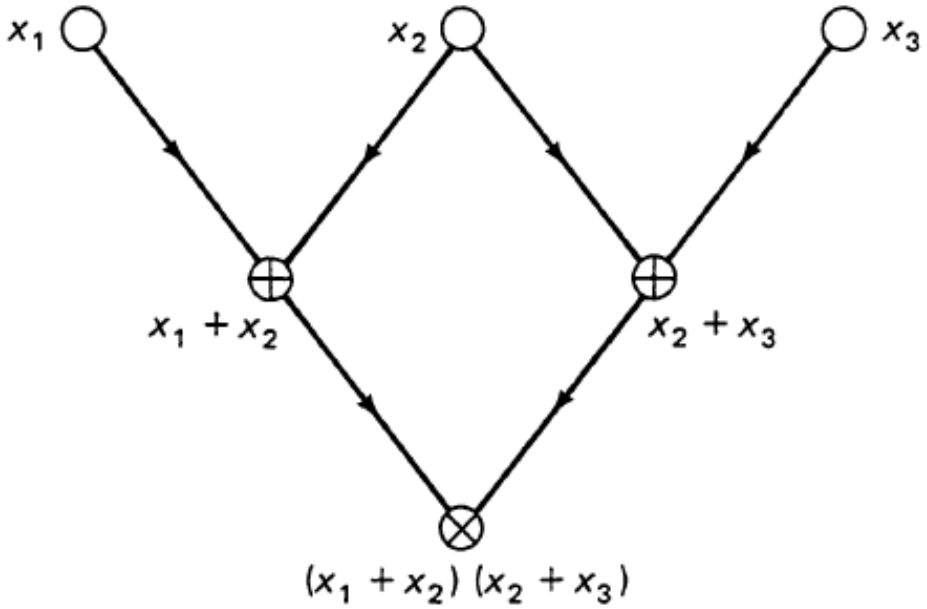


Figure 1.2.2 Another DAG representing an algorithm for evaluating the arithmetic expression $(x_1 + x_2)(x_2 + x_3)$. We have $T_1 = 3$ and $T_\infty = D = 2$. This should be contrasted with the DAG of Fig. 1.2.1 which solves the same computational problem and for which $T_1 = 7$ and $D = 3$. We conclude that the DAG given here represents a better parallel algorithm.

Comparatie calcul paralel vs. secvential

- pp alegerea unui model de calcul (ex: DAG)
- consideram problema de calcul parametrizata de n, dimensiunea problemei (nr de noduri de intrare in cazul DAG)
  - complexitatea in timp este in general dependenta de n
- pp. ca avem un algoritm paralel care foloseste p procesoare
  - p poate depinde de n
- $T_p(n)$ = timpul de rulare al algoritmului paralel
- $T^*(n)$ = timpul serial optimal (dificil de calculat, se folosesc estimari)

Acceleratia si eficienta

- **acceleratia** (**speed-up**):

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

  - descrie avantajul de viteza de executie al algoritmului paralel fata de cel mai bun algoritm secvential
  - valoarea ideala

$$S_p(n) = \mathrm{p}$$

- **eficienta**:

$$E_p(n) = \frac{S_p(n)}{p} = \frac{T^*(n)}{pT_p(n)}$$

  - fractiunea de timp utilizata efectiv de processor
  - valoarea ideala

$$E_p(n) = 1$$

- cazul ideal: disponibilitatea a p procesoare permite accelerarea calculului de p ori (niciun procesor nu e idle vreodata si nici nu executa operatii nefolositoare), *practic imposibil* !

Definitii alternative $T^*(n)$

- (1) timpul de executie al celui mai bun algoritm secvential
- (2) timpul de executie al unui algoritm secvential de benchmark
  - ex: implementare inmultire de matrici $O(n^3)$, benchmark rezonabil desi exista algoritmi cu timpi mai buni
- (3) timpul necesar unui singur procesor pentru a executa algoritmul paralel particular analizat (echivalent, un procesor simuleaza operatia paralela a p procesoare)
  - avantaj: exprima cat de bine a fost paralelizat un algoritm particular
  - dezavantaj: nu ajuta la intelegerea virtutilor algoritmului in sens absolut

Alegere $T^*(n)$ cf. celei de-a treia definitii

- pt DAG, $T^*(n) = T_1(n)$
- daca $p \leq O(\frac{T_1(n)}{T_\infty(n)})$, cf. 2.4 (b) avem

$$T_p(n) = \Theta\left(\frac{T_1(n)}{p}\right)$$

$$E_p(n) = \frac{T_1(n)}{pT_p(n)} = \Theta(1)$$

- daca $p = \Theta\left(\frac{T_1(n)}{T_\infty(n)}\right)$, cf 2.4(a) avem

$$T_p(n) = \Theta(T_\infty(n))$$

- **Concluzie**: pentru un numar de procesoare relativ redus se obtin implementari *eficiente (cel putin pentru modelul in care ignoram complexitatea comunicatiei)*

Legea lui Amdahl

- daca eficienta nu reprezinta o problema majora (cel putin pt. DAG-uri), cf concluziei anterioare, apare intrebarea:
  - poate fi accelerat un algoritm paralel oricat de mult, odata cu cresterea lui p?
- fie f fractiunea de timp consumata de sectiunile *ne-paralelizabile (inerent secventiale)* ale unui algoritm paralel, atunci

$$S_p(n) \leq \frac{1}{f + \frac{(1-f)}{p}} < \frac{1}{f}\ ,\ \forall p$$

Q: ce se intampla daca n creste si el*?*

Interpretare legea lui Amdahl

- daca redefinim *s = f, p = 1 – f* *si N = nr de procesoare (anterior p)* => definitie Amdahl:

$$S = 1 / (s + p / N)$$

unde s + p = 1

- presupunere implicita: *p* independent de *N (i.e., dimensiunea problemei e fixa)*
- presupunere nerealista in practica unde dimensiunea problemei scaleaza cu numarul de procesoare

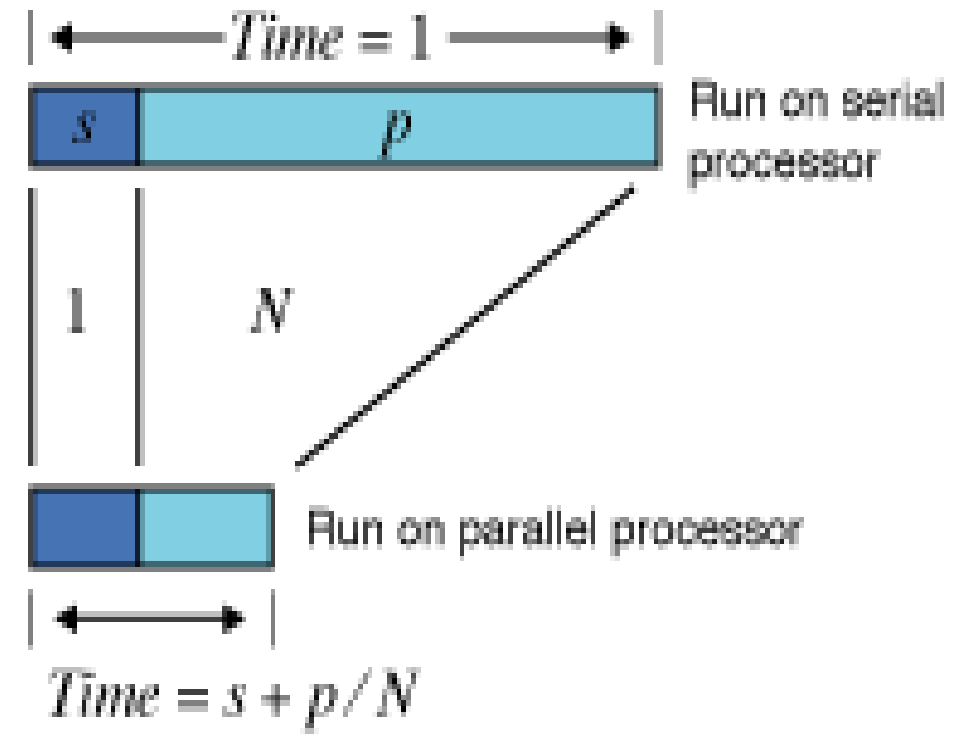


FIGURE 2a. Fixed-Size Model: *Speedup* = $1/(s + p/N)$

Legea lui Gustafson

- obs: programatorii tind sa mareasca dimensiunea problemei pe masura ce au la dispozitie un nr mai mare de procesoare
- abordare noua: pp ca timpul de rulare e constant, nu dimensiunea problemei
- acum, se considera ca p variaza liniar cu N
- obs. practica: partea seriala s (initializare vectori, incarcare program, bottleneck-uri seriale, I/O) nu creste odata cu dimensiunea problemei

Legea lui Gustafson (cont.)

- definitie noua pt speedup

$$\textit{Scaled speedup} = s + pN$$

$$= s + (1 - s)N$$

$$= N + (1 - N)s$$

- daca s e constant, dependenta de N e liniara cu panta $1 - s$
- schimbare de paradigma: scaled speedup-ul este de fapt *slowdown-ul* (incetinirea) teoretic pe care il inregistreaza un program paralel atunci cand ar rula ipotetic pe o masina cu un singur procesor

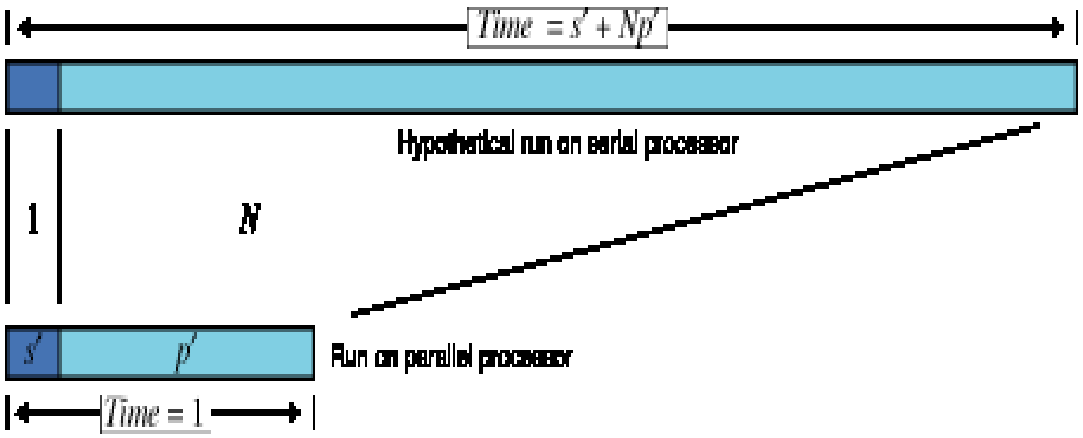


FIGURE 2b. Scaled-Size Model: Speedup = s + Np

Legea lui Gustafson, concluzii

- accentul folosirii unui nr crescut de procesoare cade pe necesitatea de a rula programe mai mari in acelasi timp daca se poate, nu pe scaderea timpului de executie al unei probleme de dimensiune fixa

- obs: nu orice problema poate fi marita arbitrar, ca atare exista si limitari ale legii lui Gustafson

Exemplu: adunare **n** scalari

- complexitate seriala: $T^*(n) = n - 1$
- complexitate paralela: ?

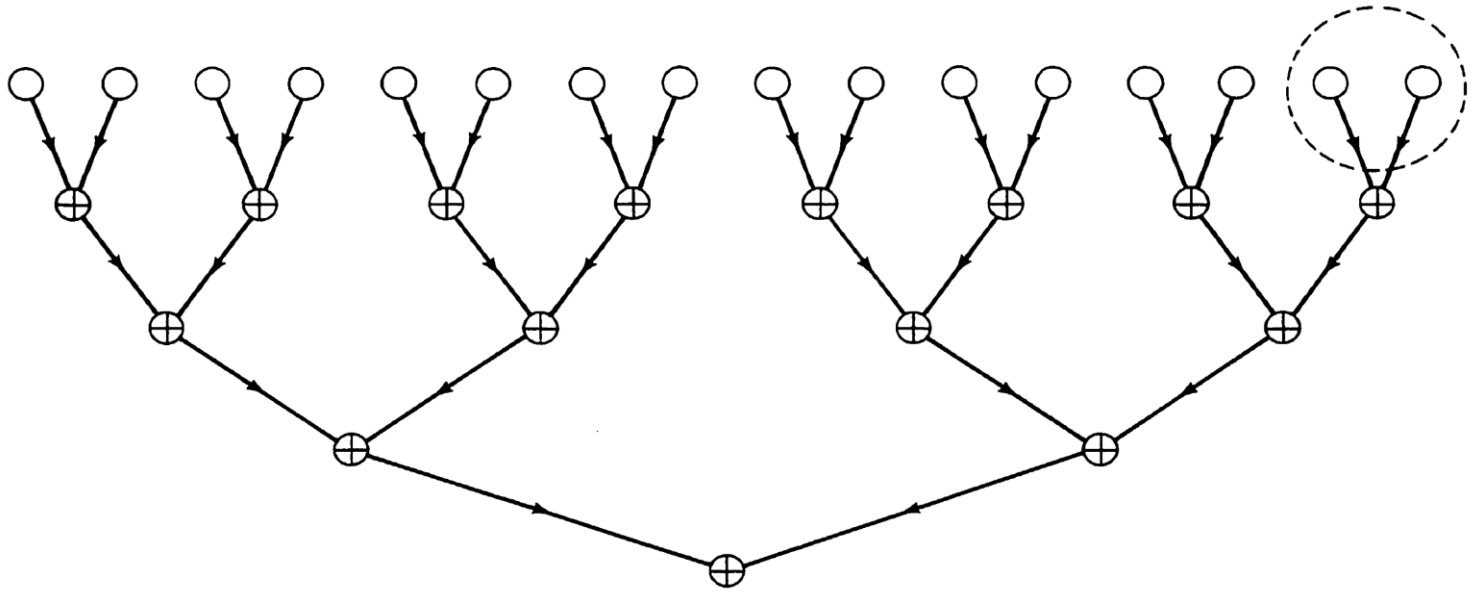


Figure 1.2.3 Parallel computation of the sum of 16 scalars. Eight processors are needed for the parallel additions at the first stage and a total of $4 = \log 16$ stages are needed. If the portion of the diagram enclosed in the dashed circle is removed, we obtain an algorithm for the parallel addition of 15 scalars. Notice that now only $7 = \lfloor 15/2 \rfloor$ processors are needed.

Adunare **n** scalari

- pp. *n* putere a lui 2
- $n/2$ *perechi (1 per procesor)*
- $n/2$ *procesoare*
- $\log(n)$ *etape*
- $T_{\frac{n}{2}}(n) = \log(n)$
- $E_{\frac{n}{2}}(n) = \frac{n-1}{\frac{n}{2}\log(n)} \approx \frac{2}{\log(n)}$

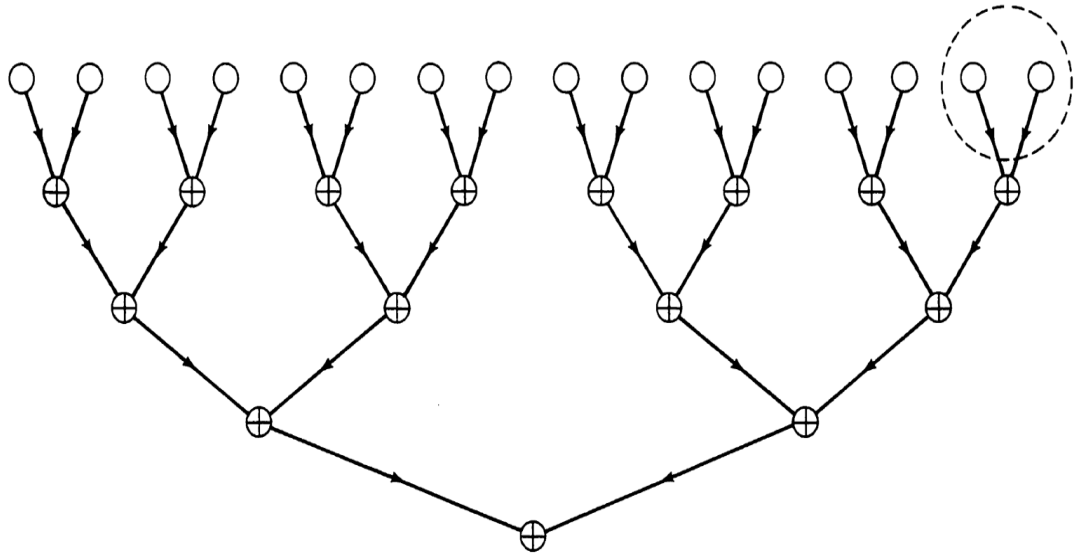


Figure 1.2.3 Parallel computation of the sum of 16 scalars. Eight processors are needed for the parallel additions at the first stage and a total of $4 = \log 16$ stages are needed. If the portion of the diagram enclosed in the dashed circle is removed, we obtain an algorithm for the parallel addition of 15 scalars. Notice that now only $7 = \lfloor 15/2 \rfloor$ processors are needed.

Adunare **n** scalari (alternativa)

- pp *log(n)* intreg, iar $\frac{n}{log(n)}$ intreg si putere a lui 2
- impartim cele n numere in $\frac{n}{\log(n)}$ grupuri de cate $log(n)$ numere
- folosim $\frac{n}{\log(n)}$ procesoare, procesorul i aduna numerele din grupul i
- timp etapa initiala: $log(n) - 1$
- raman $\frac{n}{\log(n)}$ sume partiale de adunat, folosim algoritmul anterior cu:

$$\frac{n}{2\log(n)} \textit{ procesoare}$$

=> timp de rulare: $log(n / log\, n) \leq log(n)$

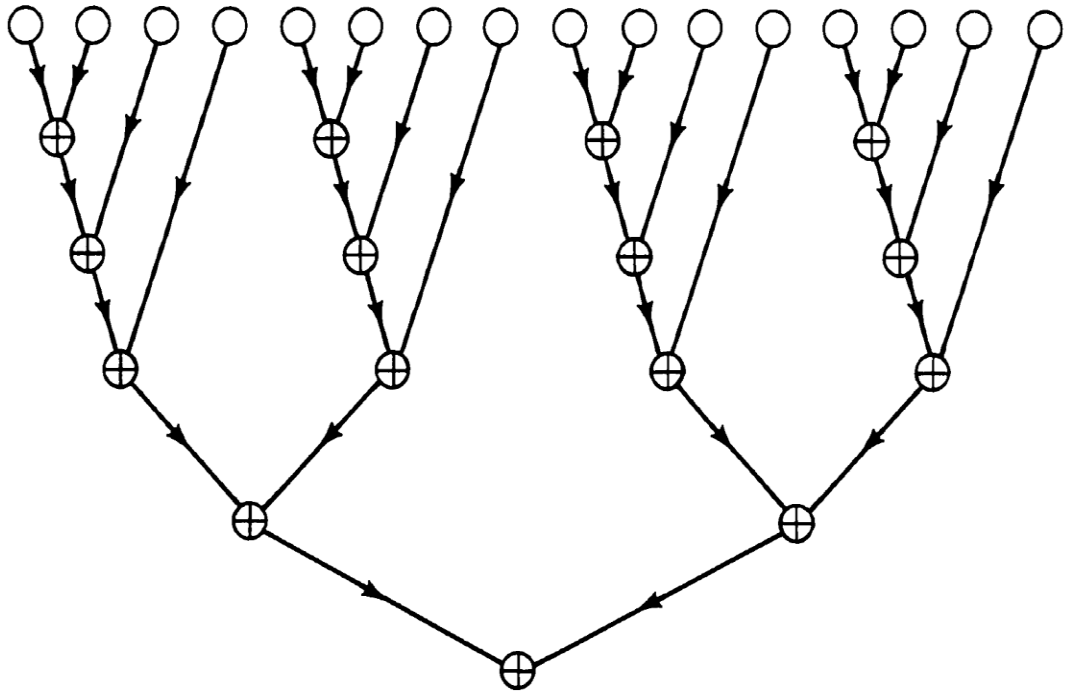


Figure 1.2.4 An alternative algorithm for the parallel addition of 16 scalars. Only four processors are used and the time requirements increase to 5 stages. Overall, however, there is an efficiency improvement over the algorithm of Fig. 1.2.3.

Concluzie algoritm alternativ

- $T_{\frac{n}{\log(n)}}(n) = 2log(n)$
  - suma celor doua etape
  - dublu fata de primul algoritm
- $E_{\frac{n}{\log(n)}}(n) = \frac{1}{2}$, semnificativ mai bine decat $\frac{2}{\log(n)}$ (pt. $\frac{n}{2}$ procesoare)
- obs: $p = \frac{n}{log(n)} \approx \frac{T_1(n)}{T_\infty(n)}$ procesoare
- alegerea unui nr de procesoare aproape egal cu $\frac{T_1(n)}{T_\infty(n)}$ imbunatateste eficienta, reconfirmare 2.4 (b)
- concluzie: imbunatatire substantiala a eficientei pt un sacrificiu mic de viteza

Exemplu: produs scalar

- x,y vectori n-dimensionali
- produs scalar

$$\sum_{i=1}^{n} x_i y_i$$

- daca se folosesc n procesoare:
  - pas 1: procesorul i calculeaza $x_i y_i$
  - apoi se foloseste algoritmul de adunare a n valori scalare

$$\Rightarrow \qquad T_n(n) = ceiling(log(n)) + 1$$

Adunare si inmultire de matrici

- n matrici de dimensiune m x m
- **adunarea** se poate face in *ceiling(log(n))* pasi cu m^2*floor(n/2)* procesoare
  - fiecare grup de *floor(n/2)* procesoare calculeaza o alta intrare a matricii suma
- **inmultirea** a doua matrici de dimensiuni m x n si respectiv n x l
  - ml produse scalare de vectori de dimensiune n
  - se poate face in *ceiling(log(n))* + *1* pasi cu nml procesoare
  - daca $n=m=l$ => n^3 procesoare si $T_1(n) = \Theta(n^3)$

  =>

$$S_{n^3}(n) = \frac{n^3}{\log n}$$

Puterile unei matrici

- fie A matrice n x n
- calculam A^k, unde $k \in N$
- pt. $k = 2^i$, $i \in N$:
  - $A^k = A^2$ x A^2 …
  - calculul dureaza $log\ k = i$ pasi, la fiecare pas avand loc o inmultire de matrici n x n

  =>

  A^k se poate calcula in timp $log\ k\ (ceiling(log\ n) + 1)$ cu n^3 procesoare
- pt $k \neq 2^i$, A^k se poate calcula in timp $\Theta(log\ k\ log\ n)$
  - *indicatie*: folositi reprezentarea binara a numarului k

=> A^2, …, A^n se pot calcula in $\Theta(log^2 n)$ cu n^4 procesoare

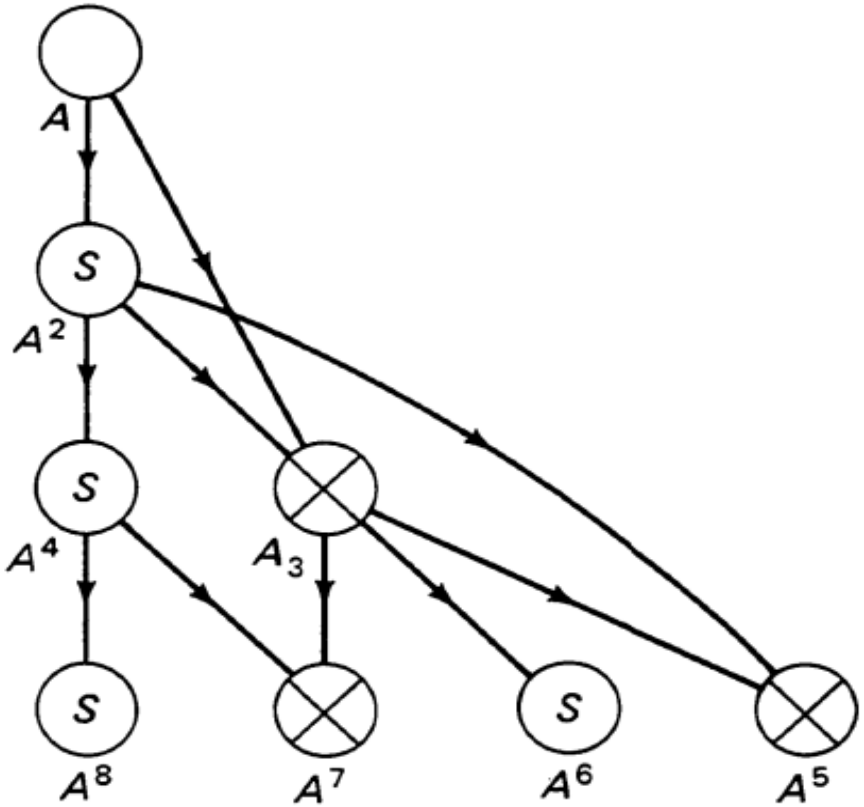
- pt fiecare A^k se foloseste un grup diferit de n^3 procesoare

Algoritm alternativ

- evita duplicarea inutila a efortului de calcul
- un nod *S* reprezinta ridicarea la patrat a unei matrici
- in pasul *k* se calculeaza puterile $2^{k-1} + 1$, …, 2^k ale matricii A

=> sunt suficiente $\Theta(\log n)$ etape pt calculul A^2, …, A^n

- fiecare etapa implica cel mult $\Theta(n)$ inmultiri simultane de matrici
  - etapa se poate calcula in $\Theta(\log n)$ cu n^4 procesoare
- => timp total de calcul $\Theta(\log^2 n)$



Analiza eficientei

- algoritmii anteriori vizeaza optimizarea timpului de rulare

=> uzual, cerere excesiva de putere de calcul (nr de procesoare) si eficienta mica

- cf. teoremei anterioare, aceiasi algoritmi se pot eficientiza daca se alege nr de procesoare $p = O(\frac{T_1(n)}{T_\infty(n)})$
- ex: inmultirea a doua matrici n x n se poate face in $\Theta(\log n)$ folosind $\Theta(n^3/\log n)$ procesoare

    => E = $\Theta(1)$

    Reducand si mai mult p => timp de executie $\Theta(\frac{T_1(n)}{p})$, cf 2.4(b)

    => doua matrici n x n se pot inmulti in $\Theta(n)$ cu n^2 procesoare si in $\Theta(n^2)$ cu n procesoare