

# Curs 9

## Programare Paralela si Distribuita

Metode de evaluare a performatei programelor paralele

Granularitate

Scalabilitate

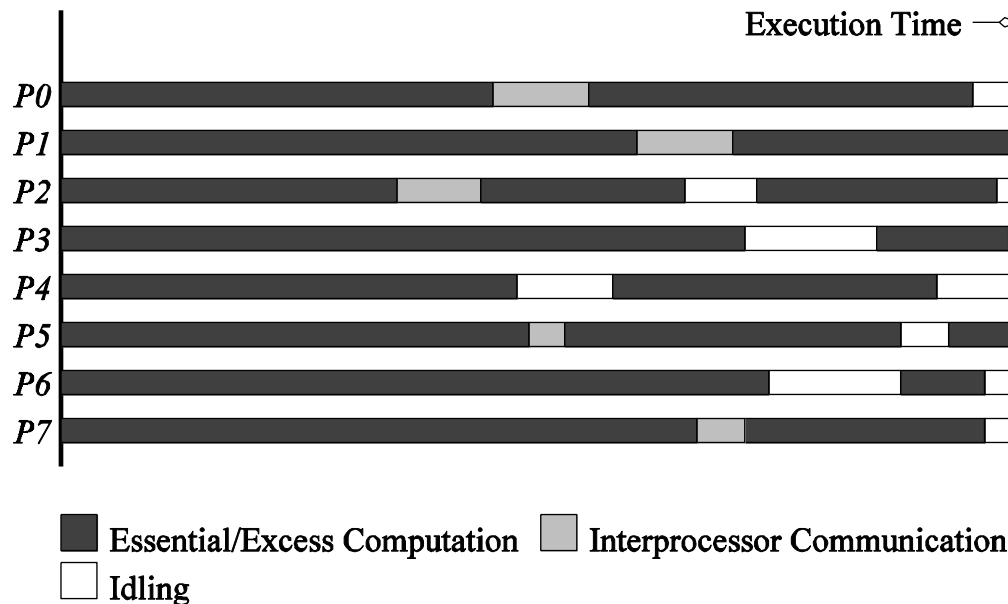
# Complexitate – consideratii generale

- Daca in cazul algoritmilor secventiali performanta este masurata in termenii complexitatilor timp si spatiu, in cazul algoritmilor paraleli se folosesc si alte masuri ale performantei, care au in vedere toate resursele folosite.
  - **Numarul de procesoare** in cazul programarii paralele =>  
o resursa importanta
- Pentru compararea corecta a variantei paralele cu cea seriala, trebuie
  - sa se precizeze arhitectura sistemului de calcul paralel  
(**sistem paralel = program + arhitectura pe care se executa**)
  - sa se aleaga algoritmul serial cel mai bun si
  - sa se indice daca exista conditionari ale performantei algoritmului datorate volumului de date.

# Observatii

- In calculul paralel, obtinerea unui timp de executie mai bun nu inseamna neaparat utilizarea unui numar minim de operatii, asa cum este in calculul serial.
- Factorul memorie nu are o importanta atat de mare in calculul paralel (relativ).
- In schimb, o resursa majora in obtinerea unei performante bune a algoritmului paralel o reprezinta numarul de procesoare folosite.
- Daca timpul de executie a unei operatii aritmetice este mult mai mare decat timpul de transfer al datelor intre doua elemente de procesare, atunci intarzierea datorata retelei este nesemnificativa, dar, in caz contrar, timpul de transfer joaca un rol important in determinarea performantei programului.

# Timp de executie



Timpul de executie al unui program paralel masoara perioada care s-a scurs intre momentul initierii primului proces si momentul cand toate procesele au fost terminate.

# Timp de executie vs Complexitate timp

- In timpul executiei fiecare procesor executa
  - operatii de calcul,
  - de comunicatie, sau
  - este in asteptare.
- Timpul total de executie se poate obtine din formula:

$$t_p = (\max i : i \in \overline{0, p-1} : T_{\text{calcul}}^i + T_{\text{comunicatie}}^i + T_{\text{asteptare}}^i)$$

- sau in cazul echilibrari perfecte ale incarcarii de calcul pe fiecare procesor din formula:

$$t_p = \frac{1}{p} \sum_0^{p-1} (T_{\text{calcul}}^i + T_{\text{comunicatie}}^i + T_{\text{asteptare}}^i)$$

# Evaluarea teoretica a complexitatii-timp

- Ca si in cazul programarii secventiale, pentru a dezvolta algoritmi paraleli eficienti trebuie sa putem face o evaluare a performantei inca din faza de proiectare a algoritmilor.
- Complexitatea timp pentru un algoritm paralel care rezolva o problema **P(n)** cu dimensiunea **n** a datelor de intrare este o functie **T** care depinde de **n**, dar si de numarul de procesoare **p** folosite.
- Pentru un algoritm paralel, un pas elementar de calcul se considera a fi o multime de operatii elementare care pot fi executate in paralel de catre o multime de procesoare.
- Complexitatea timp a unui pas elementar se poate considera a fi  $O(1)$ .
- Complexitatea timp a unui algoritm paralel este data de numararea atat a pasilor de calcul necesari dar si a pasilor de comunicatie a datelor/acces la memorie.

# Overhead

- $T_{all}$  = timpul total (insumarea timpului pentru toate elementele de procesare).
- $T_s$  = timp serial
- $T_{all} - T_s$  = timp total in care toate procesoarele sunt implicate in operatii care nu sunt strict legate de scopul problemei  
non-goal computation work  
-> total overhead.
- $T_{all} = p T_p$  ( $p$  = nr. procesoare).
- $T_o = p T_p - T_s$

## Accelerarea(“speed-up”),

- Accelerarea notata cu  $S_p$ , este definita ca raportul dintre timpul de executie al celui mai bun algoritm serial cunoscut, executat pe un calculator monoprocesor si timpul de executie al programului paralel echivalent, executat pe un sistem de calcul paralel.
- Daca se noteaza cu  $t_1$  timpul de executie al programului serial, iar  $t_p$  timpul de executie corespunzator programului paralel, atunci:

$$S_p(n) = \frac{t_1(n)}{t_p(n)}.$$

- $n$  reprezinta dimensiunea datelor de intrare,
- $p$  numarul de procesoare folosite.



# Variante

- **relativa**, cand  $t_s$  este timpul de executie al variantei paralele pe un singur procesor al sistemului paralel;
- **reala**, cand se compara timpul executiei paralele cu timpul de executie pentru varianta seriala cea mai rapida, pe un procesor al sistemului paralel;
- **absoluta**, cand se compara timpul de executie al algoritmului paralel cu timpul de executie al celui mai rapid algoritm serial, executat de procesorul serial cel mai rapid;
- **asimptotica**, cand se compara timpul de executie al celui mai bun algoritm serial cu functia de complexitate asimptotica a algoritmului paralel, in ipoteza existentei numarului necesar de procesoare;
- **relativ asimptotica**, cand se foloseste complexitatea asimptotica a algoritmului paralel executat pe un procesor.

**Analiza asimptotica** (considera dimensiunea datelor  $n$  si numarul de procesoare  $p$  foarte mari) ignora termenii de ordin mic, si este folositoare in procesul de constructie al programelor performante.

# Eficienta

- Eficienta este un parametru care masoara gradul de folosire a procesoarelor.
- Eficienta este definita ca fiind:

$$E = Sp/p$$

- Daca accelerarea este cel mult egala cu p se deduce ca valoarea eficientei este subunitara.

# Legea lui Amdahl

- Afirmă că accelerarea procesării depinde de raportul părții secvențiale față de cea paralelizabilă:

seq = fracția calculului secvențial; (e.g. 20%  $\Rightarrow$  seq=20/100)

par = fracția calculului paralelizabil; (e.g. 80%  $\Rightarrow$  par=80/100)

Se consideră **calculul serial**  $T_s = \text{seq} + \text{par} = 1$  unitate

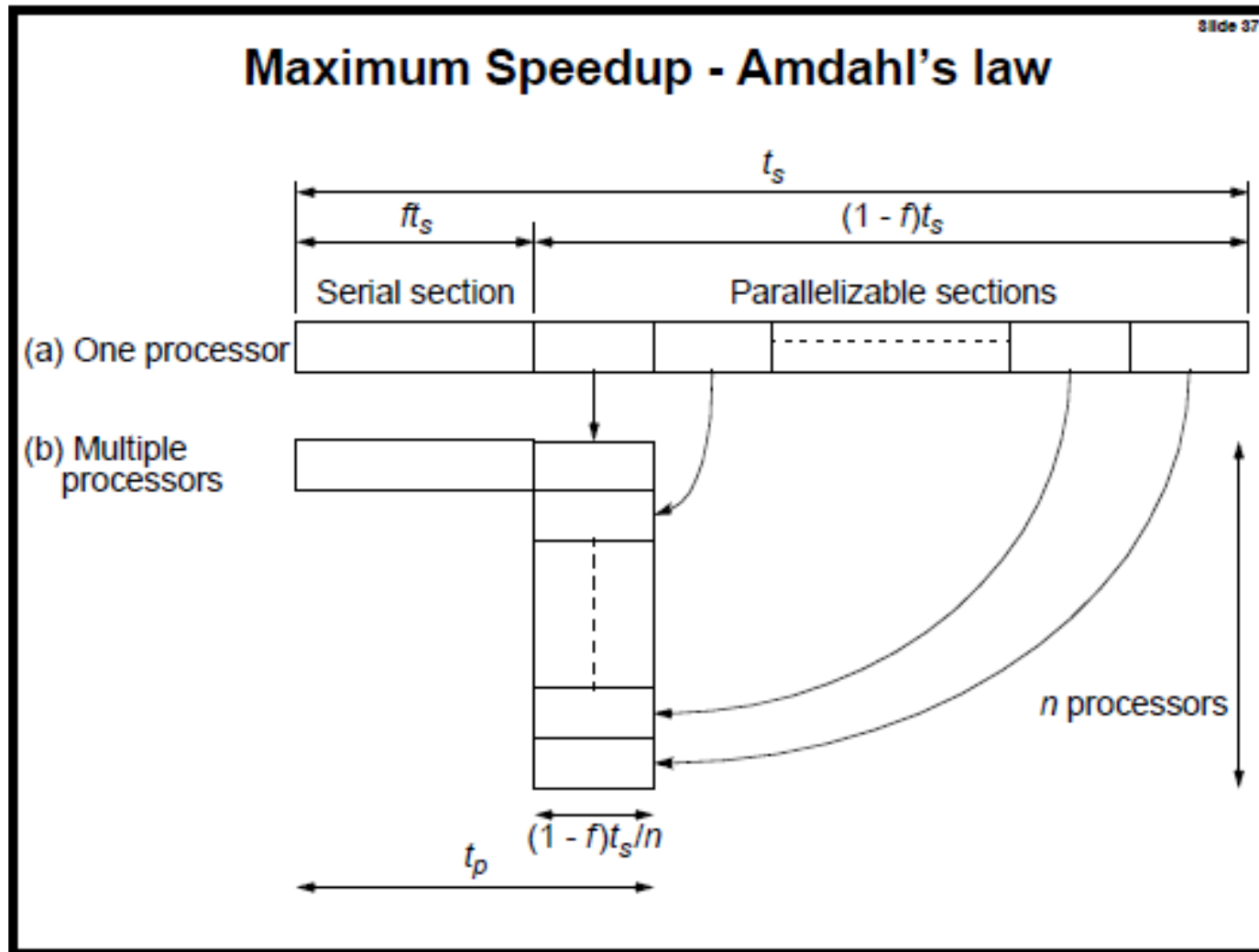
Speedup =  $1 / (\text{seq} + \text{par}/p)$

par =  $(1 - \text{seq})$ ,  $p = \#$  procesoare

- $p \rightarrow \text{infinit} \Rightarrow S \sim 1/\text{seq}$  (e.g.  $S \sim 100/20=5$ )
- Limita superioară a accelerării este dată de fracția părții secvențiale.
- ATENȚIE: Nu se face analiză în funcție de dimensiunea problemei!

Prezentare  
alternativa

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1 + (n-1)f}$$



# Legea lui Gustafson

- Considera ca atunci cand dimensiunea problemei creste, partea seriala se micsoreaza in procent :
- $m$  = dimensiunea problemei,  $p$  = # procesoare,  
     $\text{seq}(m)$  = fractia calcului secvential;  
     $\text{par}(m)$  = fractia calcului paralel;

Considerand ca **programul paralel** se executa intr-o unitate de timp :

$$T_p = \text{seq}(m) + \text{par}(m) = 1$$

$$T_s = \text{seq}(m) + p * \text{par}(m)$$

Atunci

$$\text{speedup} = T_s / T_p = \text{seq}(m) + p * \text{par}(m)$$

$$\text{speedup} = \text{seq}(m) + p(1 - \text{seq}(m))$$

Daca  $\text{seq}(m) \rightarrow 0$  atunci cand  $m \rightarrow \infty \Rightarrow$  se obtine  $\sim$  accelerare liniara.

# Legea lui Gustafson – optimista

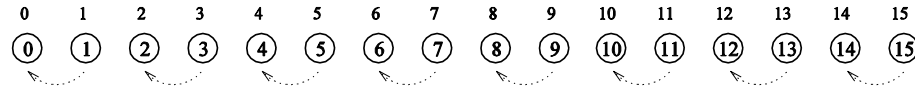
## Legea lui Amdahl - pesimista

- Legea lui Gustafson -> presupune ca partea seriala (costul ei) ramane constanta – nu creste odata cu cresterea problemei.
- Legea lui Amdahl -> presupune ca **procentul** partii secventiale este constant - nu depinde de dimensiunea problemei

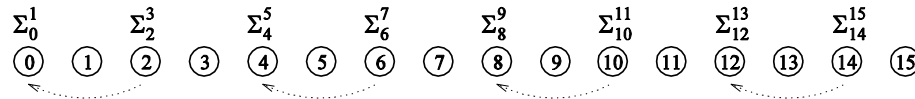
# Exemple

- Adunarea a  $n$  numere
- Daca  $n = \text{putere a lui } 2 \Rightarrow T_p = \log n$

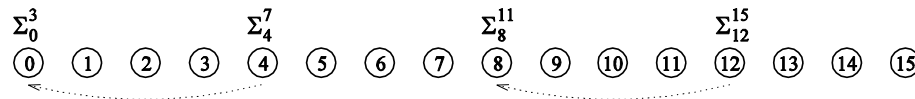
# Adunare – log n pasi



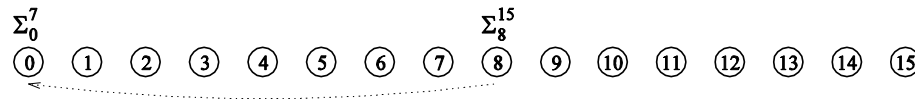
(a) Initial data distribution and the first communication step



(b) Second communication step



(c) Third communication step



(d) Fourth communication step



(e) Accumulation of the sum at processing element 0 after the final communication

$n=16$ ;  $p=16$ s ..



## Exemplu (continuare)

=>

- $t_c$  =  *timp calcul pt o operatie de adunare*
- $T_{com} = t_s + t_w$  pt o operatie de comunicatie (per word)
  - sunt  $O(n)$  operatii de comunicatie dar si operatiile de comunicatie se pot executa simultan  $T_{com} = \Theta(\log n)$

$$T_p = \Theta(\log n)$$

- Stim ca  $T_s = \Theta(n)$
- Speedup  $S = \Theta(n / \log n)$  ( $p=n \Rightarrow E = \Theta(1 / \log n)$ )

# Accelerare – superliniara?

- $S = 0$  (programul nu se termina niciodata).
- $S < p$  (teoretic)

– In caz contrar :

oricare procesor ar fi implicat in calcule pentru rezolvarea problemei un timp

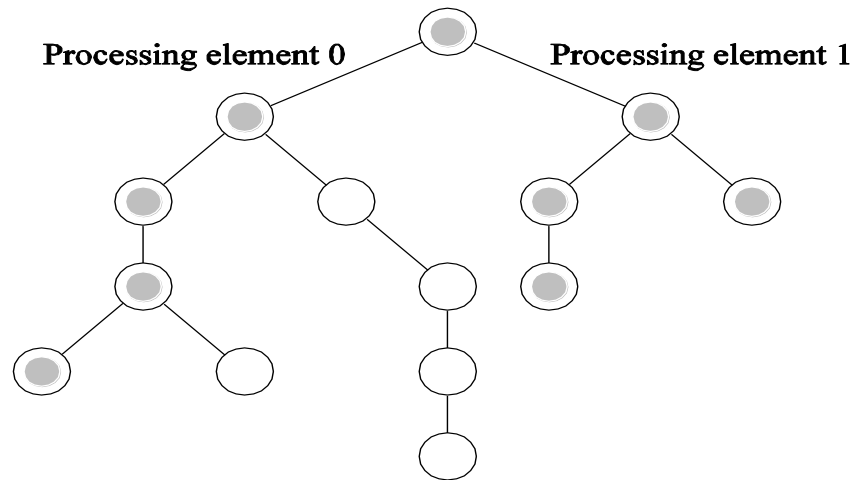
$$T < T_s / p$$

*in conditiile in care  $T_s$  depinde de numarul de operatii ( $N$ ) care se efectueaza*

$$T < (N/p)$$

*=> se efectueaza mai putine operatii in total*

# Superlinear Speedups



Cautare intr-un arbore nestructurat.

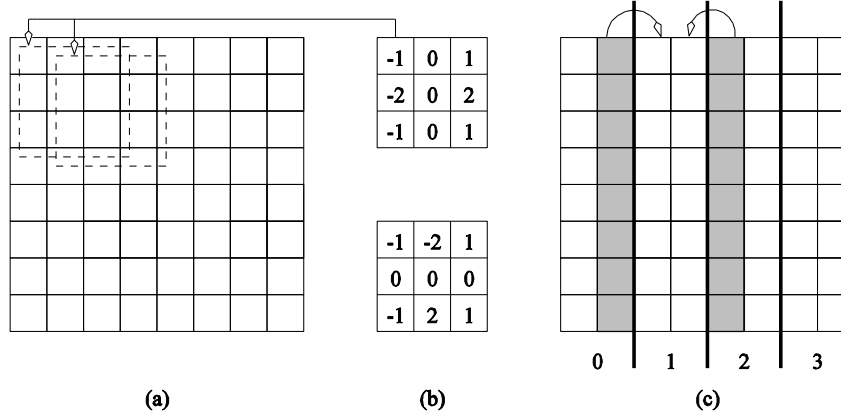
# Exemplu

Problema : *filtrare imagini = model distribuit*

- modificari pe celule de  $3 \times 3$  pixeli.

Daca o operatie aritmetica necesita pentru calcul  $t_c$ ,

timpul serial pt o imagine de  $n \times n$  este  $T_S = 9t_c n^2$ .



- Partitionare verticala =>  $n^2 / p$  pixels.
- Marginea fiecarui segment =>  $2n$  pixels.
- Nr de valori care trebuie comunicate =  $2n$  =>  $2(t_s + t_w n)$ .

*Timp de comunicare:*  $t_{com} = t_s + t_w * (message\_size)$   
 ( $t_s$  – timp de start al unei comunicatii)

*calculul executat de fiecare process:*

$$T_p^s = 9 t_c n^2 / p$$

# Evaluare metrice

- Timpul paralel:

$$T_P = 9t_c \frac{n^2}{p} + 2(t_s + t_w n)$$

- Accelerarea si eficienta:

$$S = \frac{9t_c n^2}{9t_c \frac{n^2}{p} + 2(t_s + t_w n)}$$

$$E = \frac{1}{1 + \frac{2p(t_s + t_w n)}{9t_c n^2}}.$$

# Costul

- Costul se definește ca fiind produsul dintre timpul de execuție și numărul maxim de procesoare care se folosesc.

$$C_p(n) = t_p(n) \cdot p$$

- Această definiție este justificată de faptul că orice aplicație paralelă poate fi simulată pe un sistem secvențial, situație în care unicul procesor va executa programul într-un timp egal cu  $O(C_p(n))$ .
- O aplicație paralelă este **optima** din punct de vedere al costului, dacă valoarea acestuia este egală, sau este de același ordin de mărime cu timpul celei mai bune variante secvențiale ---  $C_p = O(t_s)$ .
- aplicația este **eficientă** din punct de vedere al costului dacă  $C_p = O(t_s \log p)$ .

# Costul unui sistem paralel (algorithm + sistem)

- $\text{Cost} = p \times T_p$
- Costul reflecta suma timpului pe care fiecare procesor îl petrece în rezolvarea problemei.
- Un sistem paralel se numește optimal dacă costul rezolvării unei probleme pe un calculator paralel este asimptotic egal cu costul serial.
- $E = T_s / p T_p \Rightarrow$  pentru sisteme cost optimal  $\Rightarrow E = O(1)$ .
- $\text{Cost} \sim \text{work} \sim \text{processor-time product}$ .



# Exemplu: Adunare $n$ numere pe un model distribuit

- $T_p = (t_c + t_{com}) \log n$  (pt  $p = n$ ).

$t_c$  timpul necesar unei operatii de adunare;

$t_{com}$  - timpul necesar unei operatii de comunicare;

- $C = p T_p = O(n \log n)$
- $T_s = \Theta(n) \Rightarrow$  nu este cost optimal
- Cum se poate optimiza?
  - se micsoreaza  $p$ ;  $p = n/k$
  - se considera  $p$  segmente ( $\text{dim} = n/p$ ) pentru care se calculeaza suma secvential
  - se foloseste calculul de tip arbore pentru insumarea celor  $p$  sume locale
  - $T_p = t_c n/p + (t_c + t_{com}) \log p$
  - $C = t_c n + (t_c + t_{com}) p \log p \Rightarrow$  daca  $p \log p = O(n)$  atunci cost optimal

# Scalabilitate

- **Scalabilitatea este un parametru calitativ care caracterizeaza atat**
  - **sistemele paralele (numar de procesoare, unitati de memorie)**
  - cat si**
  - **aplicatiile paralele!**

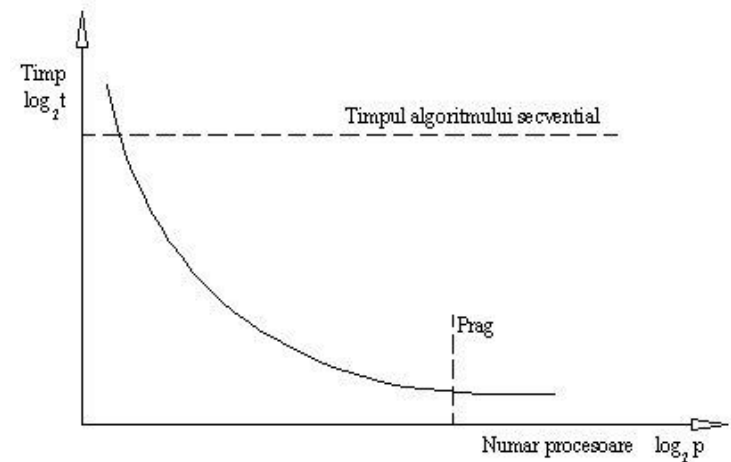
## Scalabilitatea aplicatiilor paralele

- *Un program se poate scala a.i. sa foloseasca mai multe procesoare?*
  - Adica???
  - Cum se evalueaza scalabilitatea?
  - Cum se evalueaza beneficiile aduse de scalabilitate?
- Evaluare performantei:
  - Daca se dubleaza nr de procesoare la ce ar trebui sa ne asteptam?
  - Cu cat trebuie sa creasca dimensiunea problemei daca dublam numarul de procesoare astfel incat sa obtinem aceeaasi eficienta?
  - Este scalabilitatea liniara? (raport  $p_1/p_2 = n_1/n_2$ )
- Evaluarea eficientei corespunzatoare
  - Se pastreaza eficienta pe masura ce creste dimensiunea problemei?
    - cat de mult trebuie sa creasca p?

**Scalabilitate aplicatiei: abilitatea unui program paralel sa obtina o crestere de performanta proportionala cu numarul de procesoare si dimensiunea problemei.**

# Scalabilitate

- Scalabilitatea masoara modul in care se schimba(creste) performanta unui anumit algoritm in cazul in care sunt folosite mai multe elemente de procesare.
- Un indicator important pentru aceasta este **numarul maxim de procesoare** care pot fi folosite pentru rezolvarea unei probleme.
- In cazul posibilitatii folosirii unui numar mic de procesoare, un program paralel se poate executa mult mai incet decat un program secvential.
  - Diferenta poate fi atribuita comunicatiilor si sincronizariilor care nu apar in cazul unui program secvential. (Overhead)
- Numarul minim de componente pentru o anumita partitionare, poate fi de asemenea un indicator important.



# Scalabilitate

## Definitie

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

- Analiza scalabilitatii se face pentru un **sistem (arhitectura + algoritm)**.
- *Evidentiaza cum se extrapoleaza performanta de la probleme si sisteme mici  
=>  
la probleme si configuratii mai mari.*
- Metrice pentru scalabilitate –
  - functia de isoeficianta (isoefficiency)
  - eficienta Isospeed –efficiency
  - Fractia seriala/Serial Fraction  $f$

# weak vs strong scaling analysis

- In weak scaling analysis, we evaluate the speedup, efficiency or the running time of a parallel algorithm in points  $(n, p)$  where we ensure that the problem size per processor remains constant.
  - A common practice in weak scaling analysis consists in doubling both the size of the problem and the number of processors.
  - If the running time or the efficiency remains constant, then the algorithm is scalable.
- In strong scaling analysis, we are interested in determining how far we can remain efficient given a fixed problem size.
  - Therefore, for a fixed problem size, we increase the number of processors until we observe the changes in the efficiency.

# Exemple

- Suma de 2 vectori (memorie partajata) = scalabilitate foarte buna  
n operatii independente =>  $p_{\text{maxim}} = n$

daca n creste si p poate creste => eficienta ramane la fel

Ideal

$$S = n / (n/p) = p; E = 1$$

(!!!daca se ignora timpul de creare threaduri/procese; distributia datelor!!!)

- Suma a n numere folosind p procesoare (model distribuit)

$$S = n t_c / (t_c n/p + (t_c + t_{\text{com}}) \log p) = np t_c / (t_c n + (t_c + t_{\text{com}}) p \log p)$$

$$E = n t_c / (t_o n + (t_c + t_{\text{com}}) p \log p)$$

$$p_{\text{maxim}} = n \Rightarrow E = n t_c / (t_c n + (t_c + t_{\text{com}}) n \log n)$$

# Filtru pe imagine

- Timpul paralel:

$$T_P = 9t_c \frac{n^2}{p} + 2(t_s + t_w n)$$

- Accelerarea si eficienta:

$$S = \frac{9t_c n^2}{9t_c \frac{n^2}{p} + 2(t_s + t_w n)}$$

$$E = \frac{1}{1 + \frac{2p(t_s + t_w n)}{9t_c n^2}}.$$

- Daca p creste atunci eficienta scade – cat de mult?

$$(2t_w p n / 9t_c n^2) = (2t_w p / 9t_c n) < 1$$

- $p_{\text{maxim}} = n \Rightarrow (9t_c n) / (9t_c n + 2t_s + 2nt_w) \quad (n \rightarrow \infty \Rightarrow E \sim 1)$

# Granularitate

- **Granularitatea (“grain size”)** este un parametru calitativ care caracterizeaza atat
  - sistemele paralele (numar de procesoare, unitati de memorie)
  - cat si
  - aplicatiile paralele.
- **Granularitatea aplicatiei** se defineste ca dimensiunea minima a unei unitati secventiale dintr-un program, exprimata in numar de instructiuni.
  - Prin unitate secventiala se intelege o parte din program in care nu au loc operatii de sincronizare sau comunicare cu alte procese.
- Fiecare flux de instructiuni are o anumita granularitate.
- Granularitatea unui algoritm poate fi aproximata ca fiind raportul dintre **timpul total calcul si timpul total de comunicare.**



# Granularitatea sistemului

- Pentru un **sistem paralel** dat, exista o valoare minima a granularitatii aplicatiei, sub care performanta scade semnificativ. Aceasta valoare de prag este cunoscuta ca si **granularitatea sistemului** respectiv.
  - Justificarea consta in faptul ca timpul de overhead (comunicatii, sincronizari, etc.) devine comparabil cu timpul de calcul paralel.
- De dorit
  - => un calculator paralel sa aiba o granularitate mica, astfel incat sa poata executa eficient o gama larga de programe.
  - ⇒ programele paralele sa fie caracterizate de o granularitate mare, astfel incat sa poata fi executate eficient de o diversitate de sisteme.
- Exceptii: clase de aplicatii cu o valoare a granularitatii foarte mica, dar care se executa cu succes pe arhitecturi specifice.
  - aplicatiile sistolice, in care in general o operatie este urmata de o comunicare.
    - aceste aplicatii impun insa o structura a comunicatiilor foarte regulata si comunicatii doar intre noduri vecine

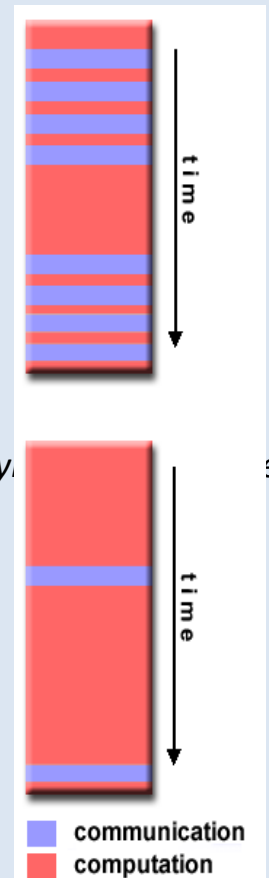
# Granularity...cont.

- **Fine-grain Parallelism:**

- Relatively small amounts of computational work are done between communication events
- Low computation to communication ratio
- Facilitates load balancing
- Implies high communication overhead and less opportunity for performance enhancement
- If granularity is too fine it is possible that the overhead required for communications and synchronization between tasks takes longer than the computation.

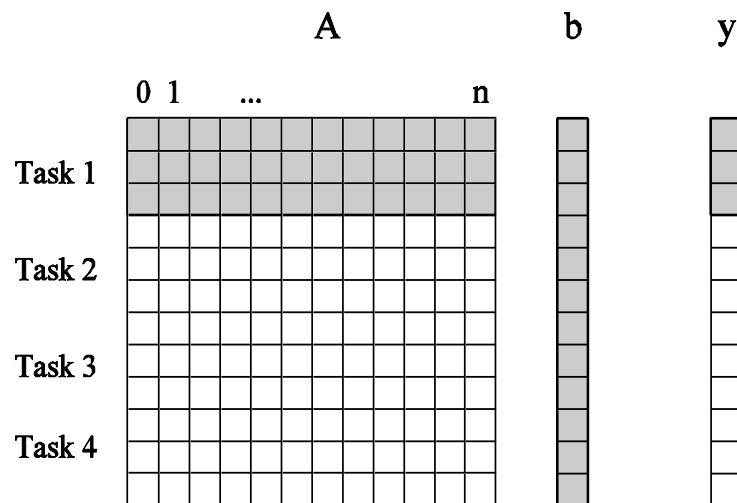
- **Coarse-grain Parallelism:**

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



# Granularitate $\Leftrightarrow$ descompunere in taskuri

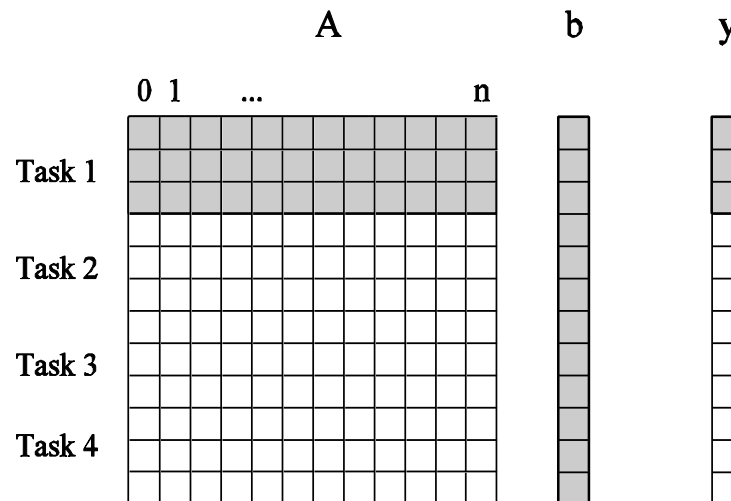
- Numarul de task-uri in care o problema se descompune **determina** granularitatea.
- numar mare  $\Rightarrow$  fine-grained decomposition
- numar mic  $\Rightarrow$  coarse grained decomposition



Exemplu: inmultire matrice

# Granularitatea decompunerii taskurilor

- Granularitatea este **determinata** (*atentie – nu definita*) de numarul de taskuri care se creeaza pt o problema.
- Mai multe taskuri => granularitate mai mica



# Efectul granularitatii asupra performantei

- De multe ori, folosirea a mai putine procesoare imbunatateste performanta sistemului parallel (ansamblu aplicatie+sistem).
- Folosind mai putine procesoare decat numarul maxim posibil se numeste ***scaling down*** (*for a parallel system*).
- Modalitatea naiva de scalare este de a considera fiecare element de procesare initiala a fi unul virtual si sa se atribuipe fiecare procesor virtual unuia real (fizic).
- Daca numarul de procesoare scade cu un factor  $n / p$ , calculul efectuat de catre fiecare procesor va creste cu acelasi factor.
- Costul comunicatiei nu creste pentru ca comunicatia intre unele procesoare virtuale se va face in cadrul aceluiasi procesor (real).

# Gradul de paralelism (*DegreeOfParallelisation*)

- DOP al unui algoritm este dat de numarul de operatii care pot fi executate simultan.
  - fina – numar mare de operatii executate in paralel
  - medie
  - Bruta

Exemplu: suma numerelor dintr-un sir=>  $DOP = n/2$

# *DOP(Degree of Parallelism)*

- *Gradul de paralelism DOP* („degree of parallelism”) =
  - (al unui program)
    - numarul de procese care se executa in paralel intr-un anumit interval de timp.
    - Numarul de operatii care se executa in paralel intr-un anumit interval de timp
  - (al unui sistem)
    - numarul de procesoare care pot fi in executie in paralel intr-un anumit interval de timp
- *Profilul paralelismului* = graficul *DOP* in functie de timp (pentru un anumit program).

Depinde de:

- structura algoritmului;
- optimizarea programului;
- utilizarea resurselor;
- conditiile de rulare.

## **In concluzie:**

- accelerarea indica castigul de viteza de calcul intr-un sistem paralel;
- eficienta masoara partea utila a prelucrarii (lucrului) efectuate de  $n$  procesoare;
- costul masoara efortul necesar in obtinerea unui viteze de calcul mai mare
- scalabilitatea masoara gradul in care o aplicatie poate folosi eficient un numar mai mare de procesoare
- granularitatea depinde de raportul intre operatiile de calcul si cele de comunicare/sincronizare



Referinte:

Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar  
``Introduction to Parallel Computing",