Curs 11

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 complexitatiilor 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,
  - sa se aleaga algoritmul serial cel mai bun si
  - sa se indice daca exista conditionari ale performantei algoritmului datorita volumului de date.

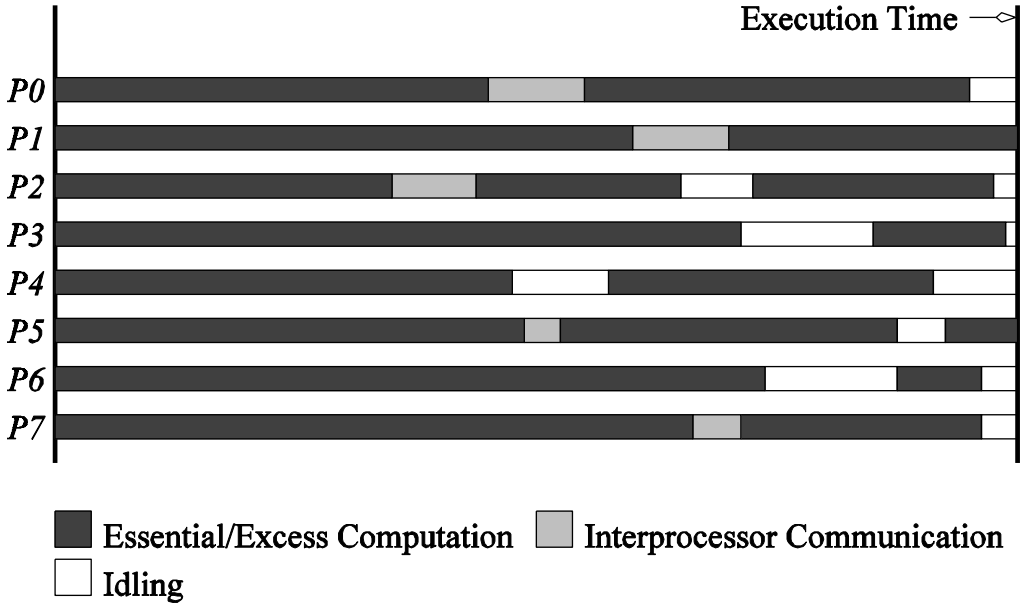
Observatii

- 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 ın calculul paralel (relativ).
- In schimb, o resursa majora ın 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 ın determinarea performantei programului.

Timp de executie



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

Complexitatea 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:

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

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

$$t_p = \frac{1}{p} \sum_{0}^{p-1} (T^i_{calcul} + T^i_{comunicatie} + T^i_{asteptare})$$

Evaluarea complexitatii

- 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 Pn 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 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.

Overhead

- T_{all} = timpul total (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).
  - nume -> *total overhead*.
- $T_{all} = p\ T_P$ (p = nr. procesoare).
- $T_o = p\ T_P - T_S$

Accelerarea(“speed-up”),

- Accelerarea notata cu Sp, 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 tp 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 ts 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, ın 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 ın procesul de constructie al programelor performante.

Eficienta

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

  $E = Sp/p$

- Se deduce ca valoarea eficientei este ıntotdeauna subunitara.

Legea lui Amdahl

- Afirma ca accelerarea procesarii depinde de procentul partii secvetiale fata de cea paraleizabila:

seq = fractia calcului secvential;
par = fractia calculului paralel;

Se considera calculul serial = 1 unitate

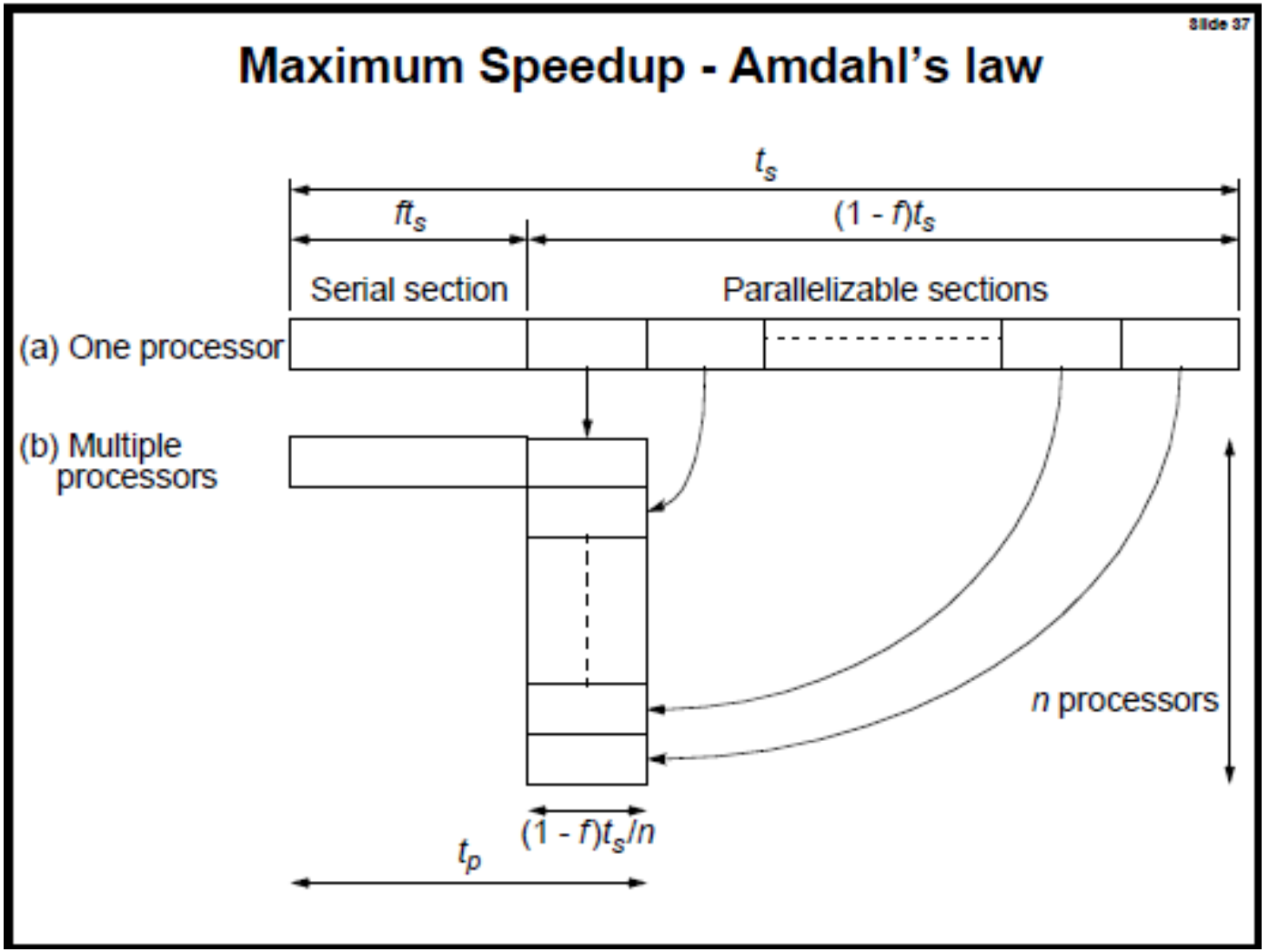
Speedup = 1/(seq + par/n)

seq = (1 – par), n= # procesoare

- *n -> infinity*, => S ~ 1/seq.
- Limita superioara a accelerarii este limitata de fractia partii secventiale.
- *Nu se face analiza in functie 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, n = # procesoare,

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

T_p = seq(m) + par(m) = 1

par(m) = 1 – seq(m), (T_s = seq(m)+n*par(m))

Atunci

- **speedup = T_s / T_p = seq(m)+n*par(m)**

speedup = seq(m) + n(1 – seq(m))

Daca seq(m)-> 0 atunci cand n->∞ -> se obtine ~ accelerare liniara.

Legea lui Gustafson – optimista
Legea lui Amdahl - pesimista

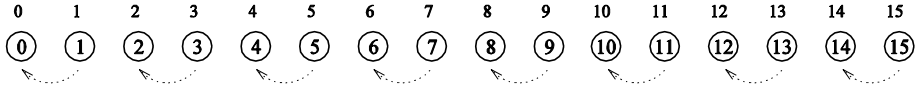
- Presupune ca partea seriala (costul ei) ramane constant – nu creste odata cu cresterea problemei.

- Legea lui Amdahl presupune o dimensiune fixa a problemek si ca partea secventiala nu depinde de numarul de procesoare.

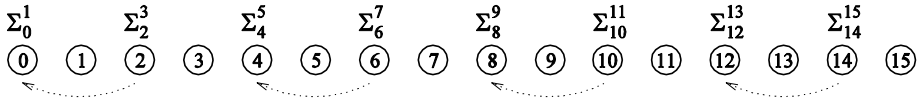
Exemple

- Adunarea a n numere
- Daca ***n*** = putere a lui 2 => T_p = **log *n***

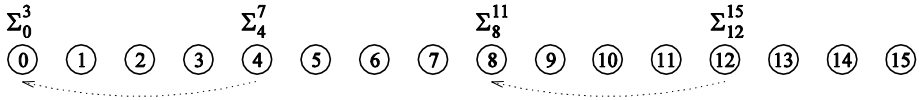
Adunare – log n pasi



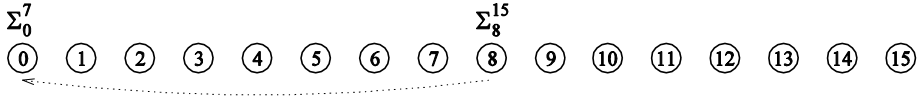
(a) Initial data distribution and the first communication step



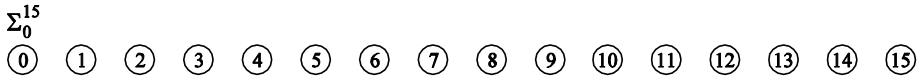
(b) Second communication step



(c) Third communication step



(d) Fourth communication step



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

n=16; p=16s . .

Exemplu (continuare)

=>

- $t\_C =$ ***timp 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 $T\_com = \Theta(\log n)$

  $T_P = \Theta(\log n)$

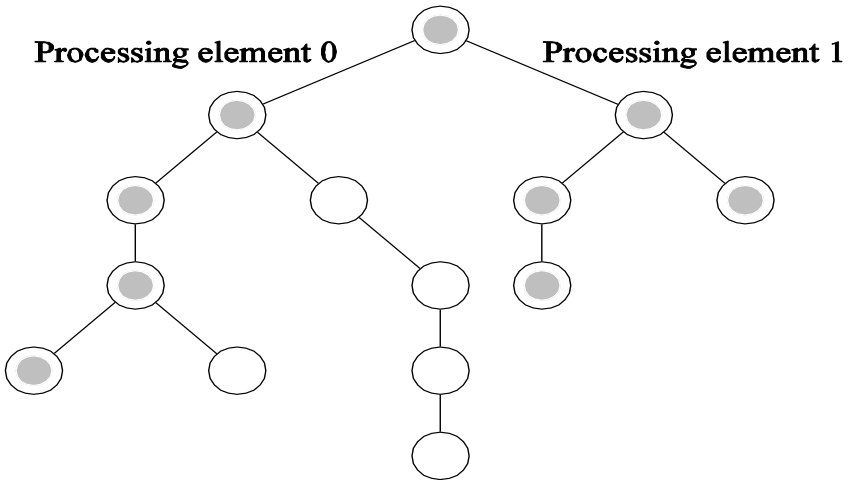
- Stim ca $T_S = \Theta(n)$

- Speedup $S = \Theta(n / \log n)$

Accelerare – superliniara?

- S= 0 (the parallel program never terminates).
- S < $\boldsymbol{p}$ (teoretic)
  - In caz contrar un procesor ar fi implicat in calcule pentru rezolvarea problemei un timp T < $\boldsymbol{T_S / p}$.
  - Se contrazice presupunerea ca se foloseste cel mai rapid program serial (in evaluare accelerare).

Superlinear Speedups



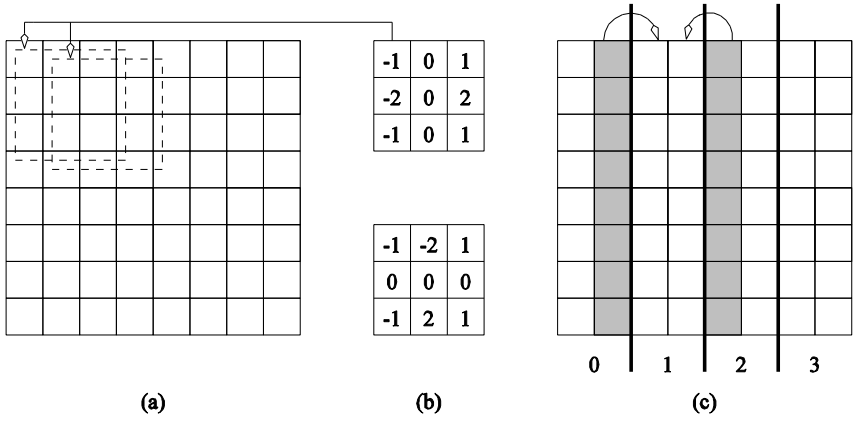
Cautare intr-un arbore nestructurat.

Exemplu

Problema : *edge-detection in images.*

Se aplica modificari pe celule de ***3*** x ***3*** pixeli.

Daca o operatie aritmetica necesita t_c,
timpul serial pt o imagine de ***n*** x ***n*** este $T_S = t_c\, n^2$.



- Partitionare veriticala => n^2 / p pixels.

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

$T^S_p = 9 t_c n^2 / p$.

Evaluare metrici

- 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 defineste ca fiind produsul dintre timpul de executie si numarul maxim de procesoare care se folosesc.

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

- Aceasta definitie este justificata de faptul ca orice aplicatie paralela poate fi simulata pe un sistem secvential, situatie in care unicul procesor va executa programul intr-un timp egal cu $O(C_p(n))$.

- O aplicatie paralela este **optima** din punct de vedere al costului, daca valoarea acestuia este egala, sau este de acelasi ordin de marime cu timpul celei mai bune variante secventiale;

- aplicatia este **eficienta** din punct de vedere al costului daca $C_p = O(t_1 \log p)$.

Costul unui sistem paralel (algoritm +sistem)

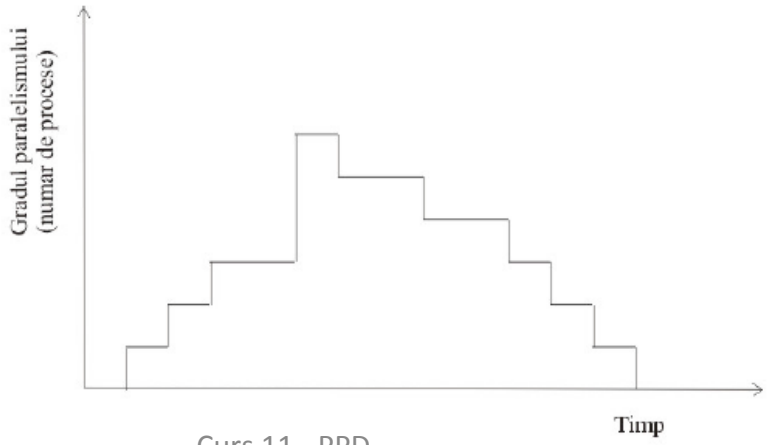
- Cost = $\boldsymbol{p}$ x $\boldsymbol{T_P}$
- Costul reflecta suma timpului pe care fiecare procesor il petrece in rezolvarea problemei.
- Un sistem paralel se numeste optimal daca costul rezolvarii unei probleme pe un calculator paralel este asimptoptic egal cu costul serial.
- $\boldsymbol{E} = \boldsymbol{T_S} / \boldsymbol{p}\ \boldsymbol{T_P}$=> pentru sisteme cost optimal => $\boldsymbol{E} = O(1)$.
- Cost ~ *work* ~ *processor-time product*.

“Work”

- volumul de lucru (“work”) – W, definit de numarul total de operatii executate de catre procesoarele active.

- Putem vedea acest volum de lucru ca si integrala profilului de paralelism al programului – adica numarul de procese active ca o functie de timp.

- Volumul de lucru poate fi evaluat si ca fiind produsul dintre dimensiunea problemei n si numarul mediu de operatii care se fac asupra unei date de intrare c: W = n*c.



Exemplu

Adunare n numere.

- $\boldsymbol{T_P} = \mathbf{log}\ \boldsymbol{n}$ (pt $\boldsymbol{p} = \boldsymbol{n}$).
- $C = \boldsymbol{p}\ \boldsymbol{T_P} = \boldsymbol{n}\ \mathbf{log}\ \boldsymbol{n}$.
- $T_s = \Theta(\boldsymbol{n})$ => nu este cost optimal.

Analiza de Impact pt cost

Consideram sortarea a **n** numere folosind **n** procesoare intr-un timp $(\log n)^2$.

- Daca timpul serial este $n \log n$ atunci

S= $n / \log n$

E= $1 / \log n$

- C = $p\,T_p = n\,(\log n)^2$.
- Nu e cost optimal.
- ? Transformare?

- Fie $p < n$, se atribuie cele n task-uri paralele la p procesoare si rezulta

$T_p = n\,(\log n)^2 / p$.

- $C = n\,(\log n)^2$ - nu e optim

- S = $p / \log n$.

Daca **n** creste accelerarea scade daca se foloseste acelasi p !

Scalabilitate

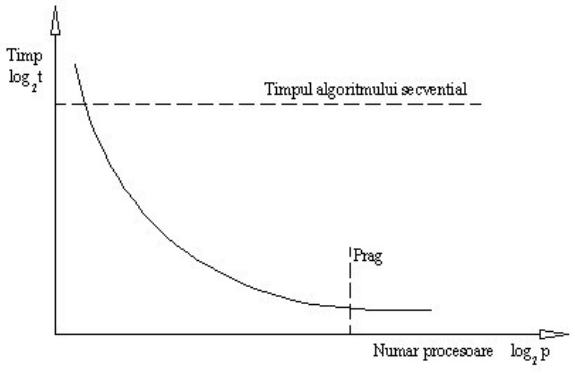
- *Un program poate scala a.i. sa foloseasca mai multe procesoare.*
  - Adica???
- Cum se evalueaza scalabilitatea?
- Cum se evalueaza benefiile aduse de scalabiltate?
- Evaluare comrativa:
  - Daca se dubleaza nr de procesoare la ce ar trebui sa ne asteptam?
  - Este scalabilitatea liniara?
- Evaluarea eficientei corespunzatoare
  - Se pastreaza eficienta pe masura ce creste dimensiunea problemei?

Definitie generala

Scalabilitate aplicatie: abilitatea unui program paralel sa obtina o crestere de performanta proportional cu numarul de procesoare si dimensiunea problemei.

Scalabilitate

- Scalabilitatea masoara modul ın care se schimba performanta unui anumit algoritm ın cazul ın 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 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 ın cazul unui program secvential. (Overhead)
- Numarul minim de componente pentru o anumita partitionare, poate fi de asemenea un indicator important.



Scalabilitate

• *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 pentru probleme si sisteme mici*
=> la probleme si configuratii mai mari.

•Metrici pentru scalabilitate –

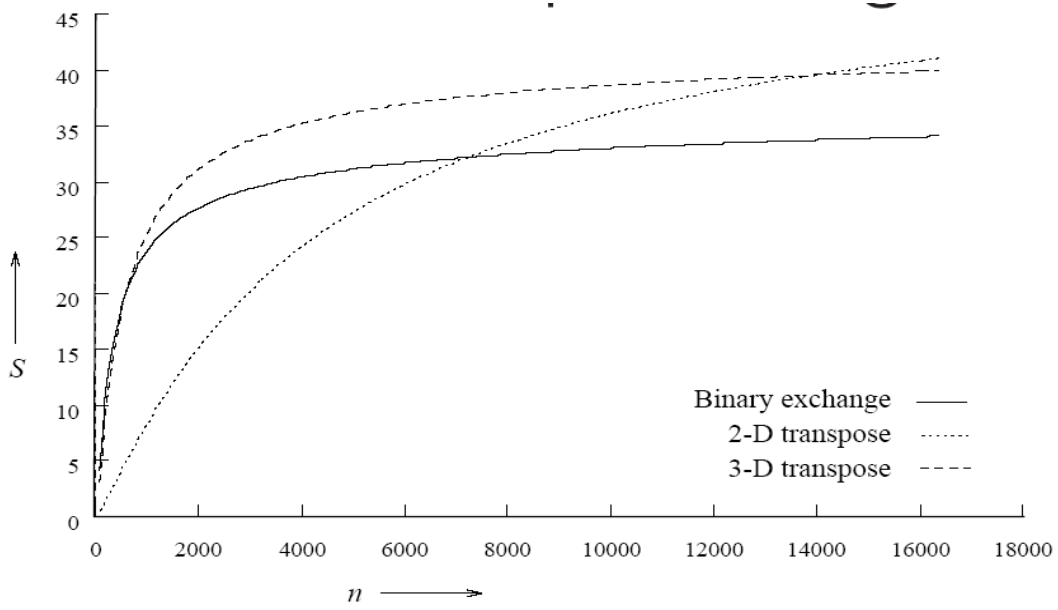
– functia de isoeficienta (isoefficiency)

– eficienta Isospeed –efficiency

– Fractia seriala/Serial Fraction *f*

Scalabilitatea sistemelor paralele

Consider three parallel algorithms for computing an ***n***-point Fast Fourier Transform (FFT) on 64 processing elements.



A comparison of the speedups obtained by the binary-exchange, 2-D transpose and 3-D transpose algorithms on 64 processing elements with $t_c = 2$, $t_w = 4$, $t_s = 25$, and $t_h = 2$.

- Este dificil sa sa evalueze caracteristicile de scalare prin observatii pe date mici si pe masini mici.

Granularitate

- ***Granularitatea("grain size") este un parametru calitativ care caracterizeaza atat***
  - ***sistemele paralele cat si***
  - ***aplicatiile paralele.***

- ***Granularitatea aplicatiei*** *se defineste ca dimensiunea minima a unei unitati secventiale dintr-un program, exprimata ın numar de instructiuni.*
  - *Prin unitate secventiala se ıntelege o parte programın care nu au loc operatii de sincronizare sau comunicare cu alte procese.*

- Fiecare flux de instructiuni are o anumita granularitate.
- Granularitatea aplicatiei se defineste ca valoarea minima a granularitatii pentru activitatile paralele ale componentelor(proces, thread, task).
- Granularitatea unui algoritm poate fi aproximata ca fiind raportul dintre timpul total calcul si timpul total de comunicare.

Granularitatea sistemului

- *Pentru un sistem 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 ın faptul ca timpul de overhead (comunicatii, sincronizari, etc.) devine comparabil cu timpul de calcul paralel.
- De dorit

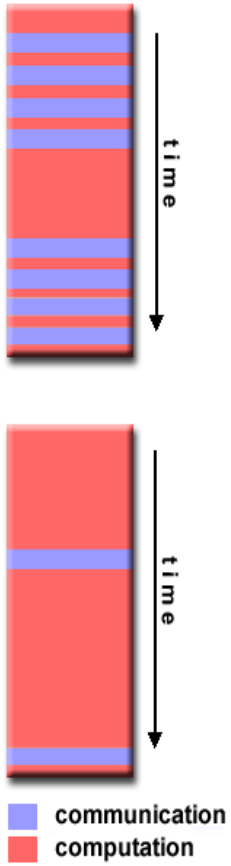
  => un calculator paralel sa aiba o granularitate mica, astfel ıncat sa poata executa eficient o gama larga de programe.

  ⇒ programele paralele sa fie caracterizate de o granularitate mare, astfel ıncat 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 comunicatie.
    - aceste aplicatii impun insa o structura a comunicatiilor foarte regulata si comunicatii doar intre noduri vecine

Granularitate…cont.

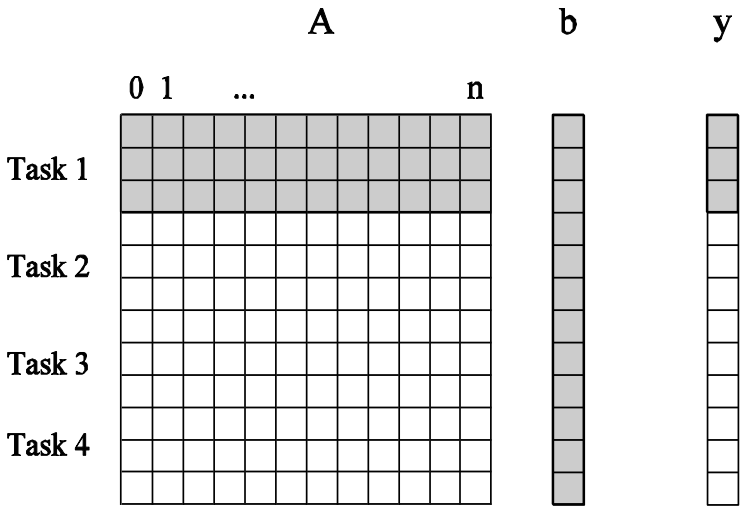
- ***Fine-grain Parallelism:***
  - *Relatively small amounts of computational work are done between communication events*
  - *Low computation to communication ratio*
  - *Facilitates load balancing*
  - *Implies high communication overhead and less opportunity for performance enhancement*
  - *If granularity 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/sy ents*
  - *High computation to communication ratio*
  - *Implies more opportunity for performance increase*
  - *Harder to load balance efficiently*

Granularitate <-> descompunere in taskuri

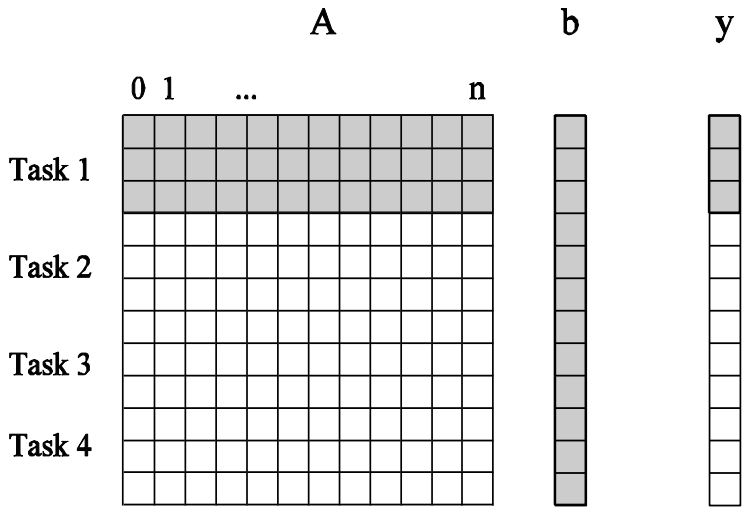
- Numarul de task-uri in care o problema se descompune determina granularitatea.
- numar mare => fine-grained
- numar mic => coarse grained decomposition



Exemplu: inmultire matrice

Granularitea decompunerii taskurilor

- Granularitatea este determinata 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 performata sistemului paralel.
- 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 initia a fi unul virtual si sa se atribuie fiecare procesor virtual unuia real (fizic).
- Daca numarul de procesoare scade cu un factor ***n*** / ***p***, calculul efectual 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).

Exemplu

- Adunare ***n*** numere cu **p** procesoare (ambele sunt puteri ale lui 2)
- Fiecarui procesor dintre cele ***p*** ii sunt atribuite ***n*** / ***p*** procesoare virtuale.

- Primii **log *n*** - **log *p*** din cei **log *n*** *pasi ai algoritmului original se simuleaza* folosind ***p*** procesoare *in* Θ *(* (***n*** / ***p***) (**log *n*** - **log *p*))=** Θ ***(***(***n*** / ***p***) **log (n/p**))

- Urmatorii **log *p*** *pasi nu necesita nici partitionare (p noduri – p procesoare)*

- T_p = Θ ((***n*** / ***p***) **log (n/p**) +**log p)**

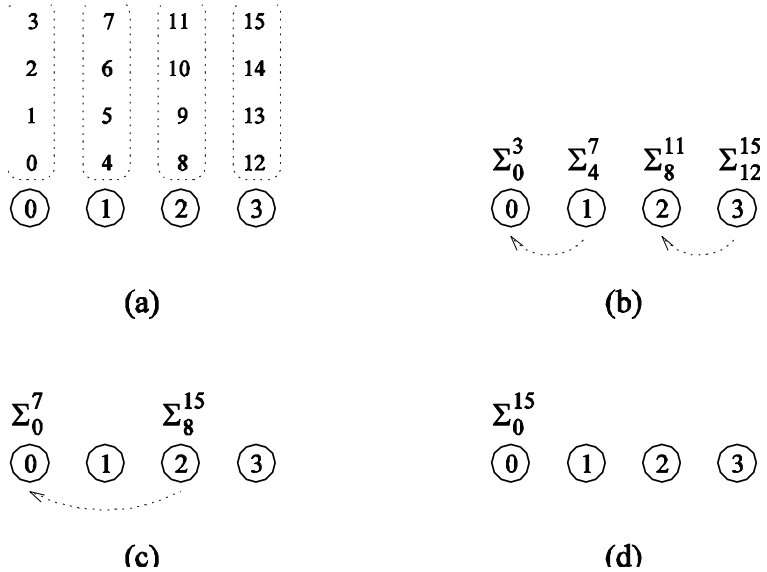
- C = O (***n*** **log *n***),
- T_s = Θ (***n***)

=> Sistemul paralel nu este cost optimal.

Varianta 2

Fiecare procesor primeste ***n*** / ***p*** numere pe care le aduna independent.

- Cele ***p*** *sume partiale pot fi adunate in timpul de* Θ(***n***/***p***).



=> program cost optimal.

Cresterea granularitatii…

$$T_P = \Theta(n/p + \log p),$$

- daca $n = \Omega(p \log p)$
- atunci sistemul este cost-optimal $\Theta(n + p \log p)$

Important − alege numarul de procese a.i. sa fie eficient calculul!

Gradul de paralelism (DOP)

- DOP al unui algoritm este dat de numarul de operatii care pot fi executate simultan.

- Similar cu granulatia sistemelor paralele
  - fina–numar mare de procesoare,
  - medie,
  - bruta–numar mic de procesoare

  se poate defini similar si DOP unui algoritm.

DOP

- *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;

-redundanta masoara dimensiunea cresterii incarcarii de lucru;

-utilizarea indica gradul de utilizare a resurselor in timpul calculului paralel;

-calitatea combina efectele accelerarii, eficientei si redundantei intr-o singura expresie pentru a evidentia meritul calculului paralel.

Referinte:

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