Metrici de performanta pentru programele paralele

Continut

- Masurarea timpului
- Modele analitice
- Timp de executie
- Surplus
- Accelerare
- Eficienta
- Cost
- Granularitate
- Scalabilitate
- Bariere
- Analiza asimptotica

Timpul

- Pentru a paraleliza un program/algoritm , trebuie cunoscuta partile din program care necesita cel mai mult timp de calcul
- Trei timpi diferiti se pot considera:

1. *Timpul pe ceas*:
   - Intervalul de timp masura pe ceas intre startul si finalizarea programului.
   - Acesta este timpul care trebuie minimizat.
2. *Timpul utilizator*:
   - Timpul la rulare efectiv utilizat de program.
   - Acesta << timpul pe ceas pt. ca programul trebuie sa astepte de ex. pentru alocarea datelor
   - Indicator pentru optimizarile necesare
3. *Timpul sistemului*:
   - Timpul utilizat nu de program insusi, ci de sistemmul de operare, ex. pentru alocarea memoriei sau acces la disc
   - Trebuie sa fie scazut.

Masurarea timpului

- Comanda Unix: **time ./shale**
  - Exemplu de iesire:
    - real 3m13.535s
    - user 3m11.298s
    - sys 0m1.915s
  - Masuoara timpul total utilizat de program
- Pentru analiza performantei, este necesara cunoastearea **timpului la rulare necesar unor parti individuale** ale programului.
  - Exista mai multe metode dependente de limbajul de programare sau sistemul de operare pentru masurarea timpului intr-un program.
    - MPI & OpenMP au propriitle functii independente de platforma pentru masurarea timpului.
      - MPI_Wtime() & omp_get_wtime() returneaza timpul pe ceas in secunde, diferenta intre rezultatele a doua apeluri ale functiei indica timpul de rulare trecut intre cele doua apeluri.
- Metode avansate de analiza a performantei: **profil.**
  - Programul insusi trebuie construit pentru a furniza informatii profilerului.
  - Exemplu: gprof
    - gprof *program* > prof.txt creeaza un fisier text cu informatia de profil.
      1. *flat profile* listeaza toate apelurile de functii/proceduri, timpul utilizat pentru ele, procentajul din timp, numarul de apeluri etc
      2. *call tree*, o lista a tuturor apelurilor de proceduri apelate de proceduri ale programului.

Catre un modelarea analitica a programelor paralele

- Un algoritm secvential este uzual evaluat prin timpul sau de executie, exprimat ca functie de dimensiunea intrarii.
- Timpul de executie a algoritmului paralel depinde nu numai de dimensiunea intrarii dar si de
  1. Numarul de elemente de procesare utilizate,
  2. Viteza lor de calcul
  3. Viteza de comunicare.
  - ⇒ Un alg. Paralel nu poate fi evaluat in izolare fata de arhitectura paralela fara a se pierde din acuratete
- Un numar de masuri ale performantei sunt intuitive:
  - Timpul pe ceas necesar pentru a rezolva o problema data pe o anumita platforma paralela .
  - Cat de repede ruleaza programul paralel relativ la programul secvential.

Timpul de executie

- **Timpul de executie secventiala** T_S a unui program
  - Este timpul scurs intre inceputul si sfarsitul executiei pe un calculator secvential.

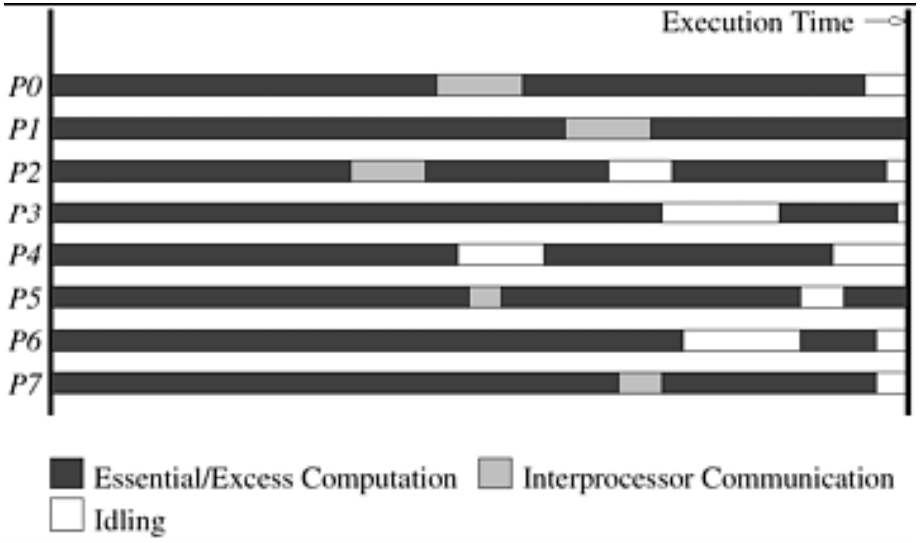
- **Timpul de executie paralela** T_P
  - Este timpul care s-a scurs din momentul in care programul a pornit pana cand toate unitatile de procesare au terminat executia.

Factorii generali care influenteaza performanta

- Algoritmul insusi trebuie paralelizat & multimea de date asupra careia se aplica trebuie sa fie astfel incat se poate aplica un numar mare de procesoare.
- Surplusurile legate de sincronizare si conflictele accesului la memorie pot conduce la deteriorarea performantei.
- Incarcarea balansata este de obicei dificila pentru a fi atinsa si lipsa acesteia conduce la deteriorarea performantei.
- Crearea algoritmilor care pot fi utilizati pe procesoare multiple conduce adesea la cresterea complexitatii computationale a algoritmilor paraleli fata de variantele secventiale.
- Divizarea datelor intre unitati de memorie diferite poate reduce conflictele de memorie si imbunatati localitatea datelor, conducand la imbunatatirea performantei.

Sursele surplusurilor in programele paralele

- Utilizand resurse hardware duble, se poate astepta ca un program sa ruleze de doua ori mai repede– Acesta este un caz foarte rar!
- Profilul de executie a unui program paralel ipotetic care este executat pe 8 elemente de procesare.
  - Profilul indica timpii petrecuti pentru a efectua calcule (esential sau in exces - necesare pentru calculul paralel), comunicare, si inactivitate.



Surse de surplus

1. ***Interactiunea intre procesoare:***
   - Orice sistem paralel netrivial solicita interactiunea intre elementele de procesare si comunicarea de date (ex. rezultate intermediare).
   - Timpul petercut pentru comunicarea datelor intre elementele de procesare este in mod uzula sursa cea mai semnificativa de surplus pentru procesarea paralela.
2. ***Inactivitate:***
   - Datorata unor fenomene diverse:
     - Incarcarea nebalansata,
     - Sincronizarea,
     - Prezenta componentelor secventiale in program.
   - Remarca 1 :
     - La generarea dinamica a sarcinilor este imposibila (sau cel putin dificila) estimarea dimenisunii subtaskurilor asignate la diverse elemente de procesare
     - ⇒ Nu se poate mentine o incarcare balansata a procesoarelor.
     - ⇒ Daca anumite procesoare au incarcari diferite, anumite elemente de procesare pot fi inactive in timp ce altele lucreaza la rezolvarea problemei.

Surse de surplus

2. ***Inactivitate:***
   - Remarca 2: elementele de procesare trebuie adesea sa se sincronizeze la un moment al executiei paralele
     - Daca elementele de procesare nu disponibile pentru sincronizare in acelasi timp, atunci cele care sunt disponibile devin inactive pana cand celelalte sunt si ele disponibile.
   - Remarca 3: Parti ale algoritmului pot sa fie ne-paralelizabilepermitand doar unei singure unitati sa lucreze (celelalte sunt intre timp inactive)

3. ***Calcul in execes:***
   - Algoritmii secventiali cei mai rapizi pot sa fie dificil/imposibil de paralelizat
     - Solutia: algoritm paralel bazat pe un algoritm secvential mai slab dar paralelizabil.
       - Adica unul cu un grad mai mare de concurenta
   - = Differenta in calculul efectuat in programul paralel fata de cel secvential.
   - Un algoritm paralel bazat pe cel mai bun algoritm paralel poate executa mai multe calcule decat algoritmul secvential.
   - Exemplu: Tranformarea Fourier rapida (ex. in procesare de imagini)
     - In varianta secventiala, rezultatele unor anumite calcule pot fi refolosite.
     - Versiunea paralela:
       - aceste rezultate nu pot fi reutilizate fiind generate de procesoare diferite
       - Astfel, anumite calcule sunt efectuate de mai multe ori pe procesoare diferite

Surplusul paralel total

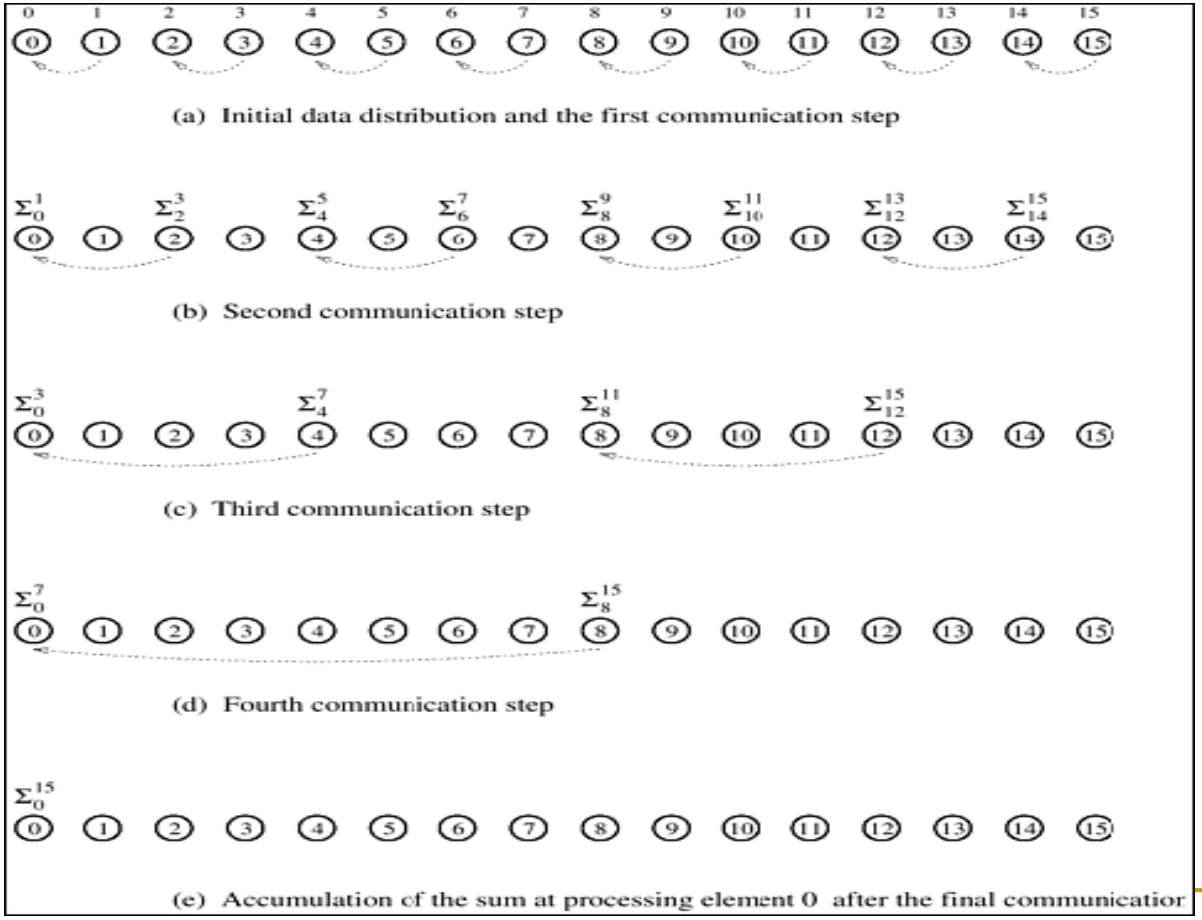
- Surplusurile intalnite in programale paralel sunt incapsulate intr-o singura expresie referita ca functie de surplus.
- Se defineste **functia surplus** sau **surplusul total al unui sistem paralel,** T_o, ca timpul total colectiv consumat de toate elementele de procesare peste cel necesitat de cel mai rapid algoritm secventia pentru rezolvarea aceleiasi probleme pe un singur element de procesare.
- Fie
  - Timpul total consumat in rezolvarea problemei de catre atoate elementele de procesare: pT_P .
  - T_S unitati din acest timp sunt consumate pentru a efectua lucru util,
  - Restul este surplusul
  - $\Rightarrow$ Functia surplus este data de $T_o = pT_P - T_S$.

Accelerare

- Intrebare: cata performanta este castigata prin paralelizarea unei aplicatii date fata de implementarea secventiala?
- Accelerarea este o masura care captureaza beneficiul relativ a rezolvarii unui probleme in paralel.
- **Accelerarea (speedup)**, S, este raportul intre timpul necesar pentru a rezolva o problema pe un singur element de procesare la timpul necesar rezolvari aceleiasi probleme pe un calculator paralel cu *p* elemente de procesare identice.
  - Acelasi tip de elemente de procesare in cazul secvential si paralel
  - accelerarea *S* este raportul timpului de executie a *celui mai bun algoritm secvential* pentru rezolvarea problemei supra timpul de executie a algoritmului paralel pentru a rezolva aceeasi problema cu *p* elemente de procesare
    - Cateodata, cel mai bun algoritm secvential pentru rezolvarea unei probleme nu este cunoscut, sau
    - Timpul sau de rulare are o constanta mare ceea ce-l face impractic pentru implementare
      - In aceste cazuri, se considera cel mai rapid algoritm cunoscut la un moment dat ca fiind cel mai bun.

Exemplu: adunarea a *n* numere intregi utilizand *n* lemente de procesare

- Daca $n = 2^k$, operatia poat fi efectuata in $\log n = k$ pasi



$T_P = O(\log n)$.
$T_S = O(n)$
$S = O(n/\log n)$.

Exemplu: sortare

- Fie exemplu paralelizarii metodei bulelor.
- Presupunem:
  - Versiunea secventiala pt sortearea a 10^5 intregi necesita 150 s
  - Un quicksort secventual poate sorta aceeasi lista in 30 s.
  - O versiune paralela a metodei bulelor, numita sortare par-impar, necesita 40 secunde pe 4 PEuri:
- Ar parea ca algortmul paralel par-impar are o accelarare de 150/40=3.75.
- Aceasta afirmatie nu este corecta!
- Accelerarea algoritmului paralel este 30/40 = 0.75 relativ la cel mai bun algoritm secvential.

Teoretic S<=p

- Daca cel mai bun algoritm secvential necesita T_S unitati de timp pentru o problema data pe o singura unitate de procesare, atunci o accelerare *S* de *p* poate fi obtinuta cu *p* elemente daca nici una dintre aceste elemente nu consuma mai mult timp decat T_S /p.
- Presupunem ca *S*>*p*

⇒ posibil numai daca fiecare element consuma mai putin decat T_S /p pentru rezolvarea problemei

⇒ o singura PE poate emula cele *p* PEuri si rezolva problema in mai putin decat T_S unitati de timp

⇒ Acest fapt este in contradictie deoarece *S* este calcculat relativ la cel mai bun algoritm secvential.

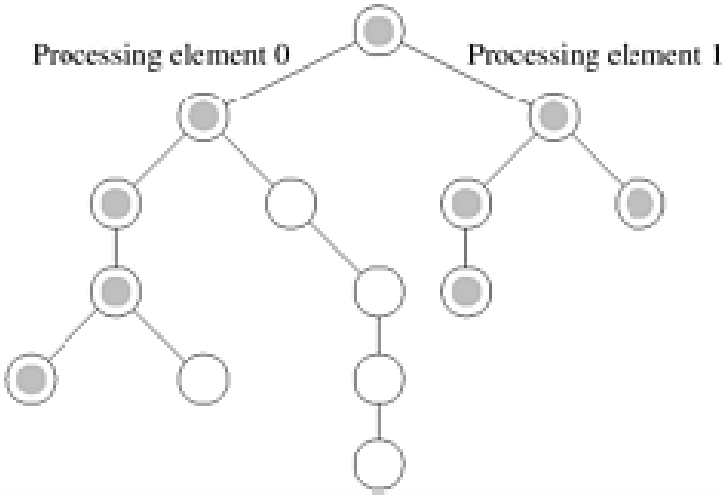
Accelerare superliniara

- In practica, o accelerare mai mare decat *p* este uneori observata.
- Uzual acest lucru se intampla daca:

1. Lucrul efectuat de algoritmul secvential este mai mare decat cel in formulare paralela
   - Exemplu: cautare
2. Datorita facilitatilor hardware implemetarea secventiala este pusa in dezavantaj.
   - De exemplu:
     - Datele pentru o problema pot fi prea mari pentru a incapea in memoria cache a unui singur elemente de procesare,
     - ⇒ Degradarea performantei sale datorita utilizarii unor elemente de memorie mai lente
     - ⇒ Cand datele sunt partitionate intre mai multe PEuri, partitiile individuale de date pot fi suficient de mici pentru a incapea in memoria cache a respectivelor elemente de procesare

Superliniaritate datorate descompunerii exploratorii

- Fie
  - Un algoritm care exloreaza nodurile frunza a unui arbore nestructurat
  - Fiecare frunza are o eticheta care ii este asociata
  - Obiectivul este acela de a gasi nodul cu eticheta data.
  - Doua elemente de procesare utilizeaza traversarea in adancime.
  - Procesorul 0 cauta in subarborele stand
  - Procesorul 1 cauta in subarborele drept
  - Sunt tratate numai nodurile umbrite din figura inainte de a gasi solutia.
  - Formularea secventiala trebuie sa parcurga intregul arbore.
- Agoritmul secvential efectueaza mai multe operatii decat cel paralel.



Eficienta

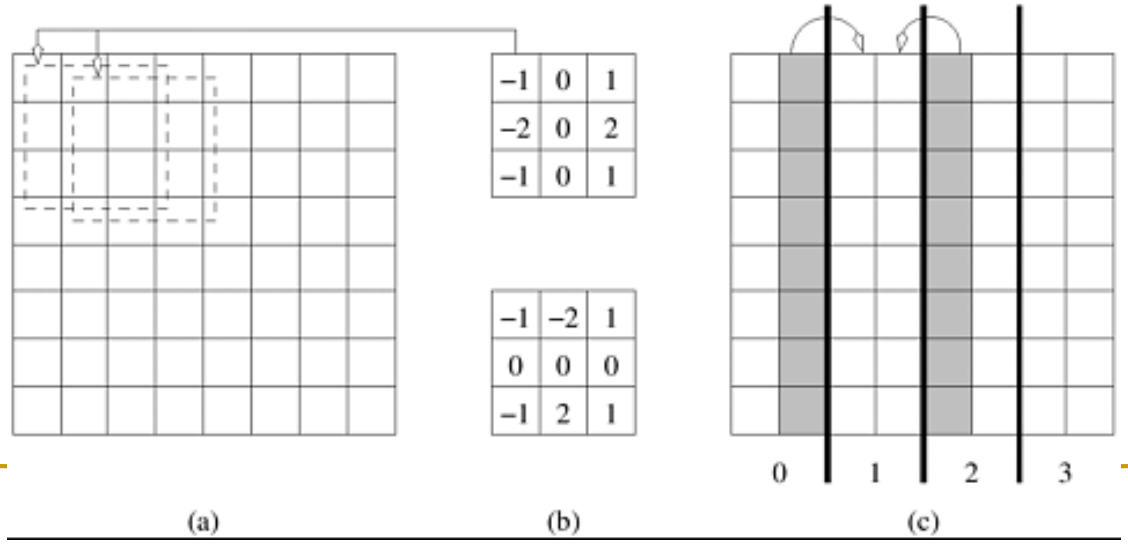
- Comportarea ideala nu este atinsa deoarece la executia unui algoritm paralel elementele de procesare nu pot dedica 100% din timpul lor calculelor algoritmului.
  - Exemplu: parte a timpului cerut de PEuri pentru a calcula suma a n numere este consumata pein inactivitate (si comunicare in sisstemele reale).
- Eficienta este o masura a fractiei de timp pentru care un element de procesare este angajat in lucru util.
- ***E* este raportul intre *S* si numarul de PEuri : *E=S/p*.**
- In sisteme paralele ideala eficienta este unu.
- In practica, eficienta este intre zero si unu

Exemple

- *Eficienta adunarii a n numere cu n elemente de procesare:* E= O(*n*/ log *n*)/*n*=O(1/ log *n*).
- Detectia muchiilor in imagini
  - Secvential:
    - Data fiind o imagine de *n* x *n* pixeli, problema detectarii muchiilor corespunde cu aplicarea a unui sablon 3x 3 la fiecare pixel.
      - Se multiplica valorile pixelilor cu valorile din sablon si se insumeaza (operatie de convolutie).
      - Se efectueaza 9 operatii multiplicare-adunare per fiecare pixel,
      - Presupunem ca fiecare sarcina de multiplicare-adunare necesita timpul t_c,
      - Atunci intreaga operatie necesita $9t_c n^2$ pe un calculator serial.

Detectarea muchiilor - paralel

- Se partitioneaza imagina in mod egal intre PEuri
- Fiecare PE aplica sablonul la sub-imaginea sa
- Pentru aplicarea sablonului la pixelii de frontiera, un PE trebuie sa obtina data care este asignata la un PE adjunct.
- Daca unui PE ii este asignata o felie verticala de dimensiune n x (n/p),
  - Trebuie sa accesesze un strat de n pixeli de la PEul din stanfa si similar la dreapta
- Algoritmul se executa in doi pasi:
  1. Schimbul de straturi de n pixeli intre fiecare doua PEuri adjuncte - $2(t_s + t_w n)$.
  2. Aplica sablonul la subimaginea locala: $9t_c n^2/p$
- Timpul total pentru algoritm este dat de : $T_P = 9t_c n^2/p + 2(t_s + t_w n)$.
- $S = 9t_c n^2 / [9t_c n^2/p + 2(t_s + t_w n)]$, $E = 1/ [1 + 2(t_s + t_w n)p / 9t_c n^2]$.



Cost

- = Timpul de rulare in paralel x numarul de PEuri utilizat
- Costul reflecta suma timpilor pe care fiecare PE il consuma rezolvand problema
  - ⇒ *E* poate fi exprimat ca raportul intre timpul de xeceutie al celui mai bun algoritm secvential pentru a rezolva problema si costul rezolvarii aceleiasi problemele pe *p* elemente de procesare.
- p=1: Costul pentru rezolvarea problemei pe un singur element de procesare este timpul de executie al celui mai rapid algorim secvential ce este cunoscut.
- Un alg. paralel se spune ca este ***optimal in cost*** (sau cost-optimal) daca costul rezolvarii problemei pe un sistem paralel are aceeasi crestere asimptotica (in termeni de O) ca si fuctie de dimensiunea intrarii ca si cel mai rapid algoritm secvential pe un singur element de procesare.
  - ⇒ Deoarece eficienta este raportul intre costul secvential la cel paralel, un alg. paralel cost-optimal are eficienta de O(1).

Exemple

- ***Costul adunarii a n numere pe n elemente de procesare.***
  - Cost= O(n log n).
  - Timpul de executie secvential este O(n) => algoritmul nu este cost-optimal.
- ***Algoritmi de sortare.***
  - Fie un algoritm de sortare care utilizeaza n elemente de procesare pentru a sorta lista in timpul $(\log n)^2$.
  - Deoarece timpul de timpul secvential este de ordinul n log n, accelerarea si eficienta acestui algoritm sunt date de n/log n si1/log n, respectiv.
  - Cost= $n(\log n)^2$ => algoritmul nu este cost-optimal

Efectul granularitatii asupra teorei performantei

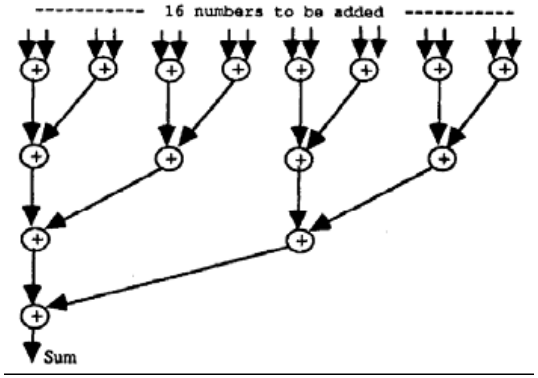
- Adunarea a n numere cu n PEuri – sunt prea multe elemente de procesare.
- In practica, asignare de bucati mari de date de intrare la PEuri.
  - Aceasta corespunde cu cresterea granularitatii de calcul pe PEuri.
- Utilizarea unui numar mai mic decat maximul posibil de nr. de PEuri pentru executarea unui alg. paralel este numita **scalare in jos** in termeni de numar de PEuri.
- O **modalitate naiva** de a scala un sistem paralel este de a construi un algoritm paralel pentru un element de intrare per PE, si apoi utilizarea mai putinor PEuri pentru a simula un numar mare de PEuri.
  - Daca sunt *n* intrari si numai *p* PEuri ($p < n$), se poate utiliza alg. Paralel construit pentru *n* PEuri prin asugnarea a *n* PEuri virtuale fiecare dintre cele *p* PEuri fizice simuland *n/p* PEuri virtuale.
  - Timpul total de rulare creste cu un factor cel mult *n/p*, iar produsul procesor-time nu creste.

    => Daca un alg. par cu *n* PEuri este cost-optimal, utilizand *p* PEs (unde $p < n$) pentru a simula *n* PEuri pastreaza cost-optimalitatea.

Efectul granularitatii asupra teorei performantei - exemplu

- Adunare: Fie $p << n$.
  - Asignare n sarcini la $p < n$ PEs => un timp parallel mai mic decat $n(\log n)^2/p$.
  - Accelerarea este $p/\log n$.
  - Exemplee:
    - sortare1024 numere (n = 1024, log n = 10) cu p=32 PEuri=> S=3.2.
    - $n = 10^6$, log n = 20 => S=1.6. Mai rau!

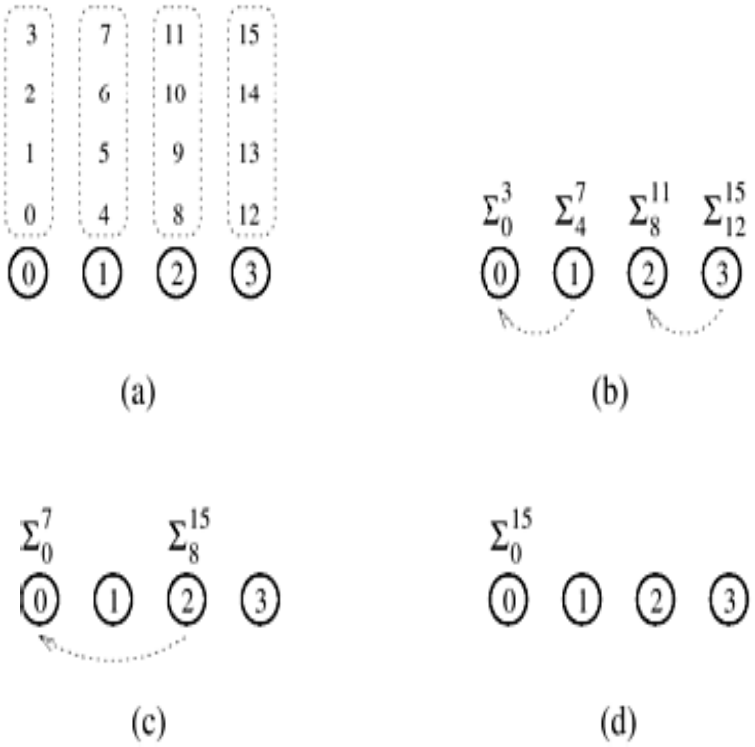
  Remarca: daca un alg paralel nu este cost-optimal ramane neoptimal in cost dupa ce granularitatea creste



n=16, p=8

Adunarea a n numere cost-optimala

- Exemplu: $n = 16$ and $p = 4$.
- In primul pas fiecare PE aduna cele n/p numere locale in timp $\Theta(n/p)$.
- Problema este redusa la adunarea la p sume partiale pe p PEuri, ceea ce poate fi realizat in $\Theta(\log p)$ prin metoda descrisa in ex. anterior.
- Timpul paralel este $TP=\Theta(n/p+\log p)$
- Costul este $\Theta(n + p \log p)$.
- Daca $n = \Omega(p \log p)$, costul este $\Theta(n)$, acelasi ca in cazul serial.
- Deci s-a obtinut cost-optimalitatea.
- Demonstrat ca maniera in care calculul este mapat pe PEuri poate determina cost-optimalitatea.
- Nota: Nu se pot face toti alg. ne-cost-optimali, cost-optimali prin scalare in jos a nr. de PEs.



Caracteristicile de scalarea a progr. paralele

- $E = 1/(1+T_O/T_S)$,
  - T_o creste cel putin linieiar cu p.
  - ⇒ Eficienta prog. Paralel scade pentru o marime a problemei (T_S constant)

Exemplu:

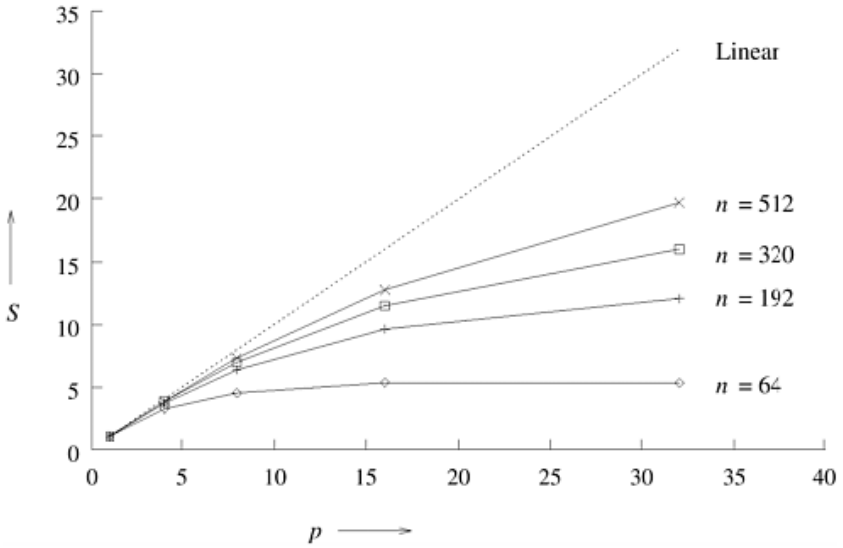
adunare n numere pe p PEuri,

$T_P = n/p + 2 \log p$,

$S = n/(n/p + 2 \log p)$,

$E = 1/(1 + 2p \log p / n)$.

Calculeaza S si E ca functii de $(n,p$



Algoritmi scalabili

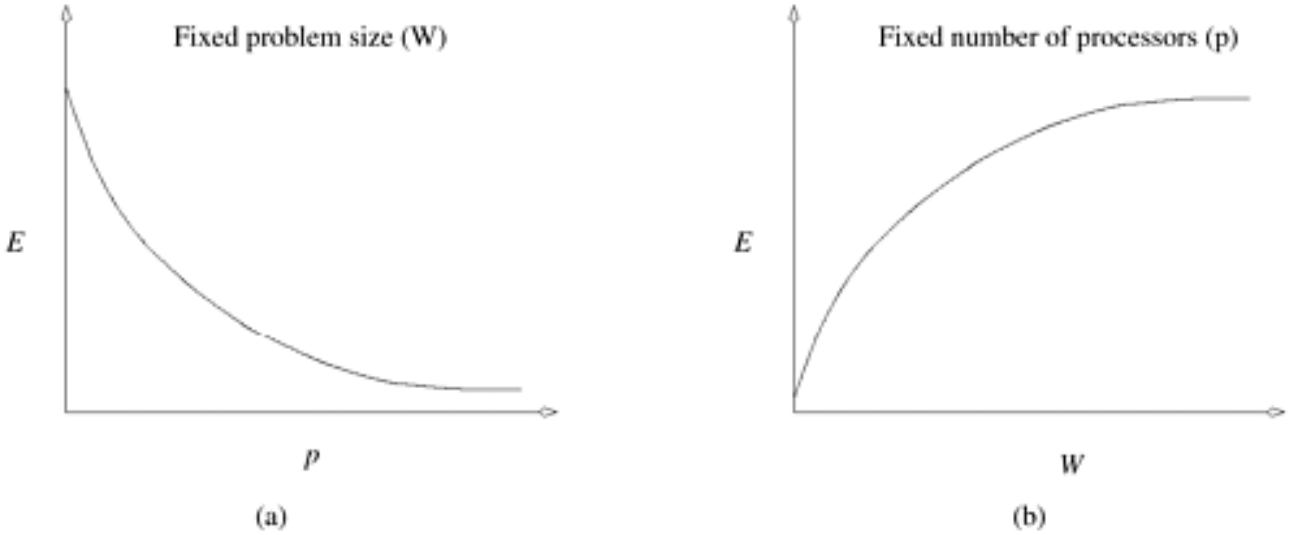
- Scalabilitatea este capacitatea de a creste *S* in proportie cu numarul de PEuri.
- Reflecta abilitatea de utilizare efectiva a cresterii resurselor.

- Exemplu + *n* numere pe *p* PE cost-optimal - $n = O(p \log p)$.
  - E=0.80 pt. $n = 64$ si $p = 4$, $n = 8 p \log p$.
  - $p = 8$, $n = 8 p \log p = 192$, E=0.80
  - $p = 16$, $n = 8 p \log p = 512$, E=0.80 .

  => Alg. paralel ramane cost-optimal la o eficienta de 0.80 daca $n = 8 p \log p$ si *p* creste.

Observatii

- Pentru o problema de dimensiune data, la cresterea numarului de PEuri, eficienta scade (Fig (a)).
- In numeroase cazuri, eficienta creste cu dimensiunea problemei (Fig (b)) cand nr. de PEuri este constant.
- **Un alg. scalabil**= este unul in care eficienta poate fi mentinuta constanta odata cu cresterea numarului de PEuri in conditiile in care si dimensiunea problemei creste



Functia de izoeficienta

- Dimensiunea problemei: W.
- Pp. ca este necesare o unitate de timp pentru a executa un pas de calcul de baza a unui algoritm

$W = T_S$ (al celui mai rapid alg. cunoscut).

$T_P=[W+T_0(W,p)]/p$, $S= W/ T_P = Wp/[W+T_0 (W,p)]$, $E=S/p= W/[W+T_0(W,p)]= 1/ [1+T_0 (W,p)/W]$.

⇒ W trebuie sa creasca odata cu p pentru a mentine E fix

- Un alg. paralel este **puternic scalabil** daca W creste liniar odata cu p
- Un alg. paralel este **slab scalabil** daca de ex. W trebuie sa creasca exponential cu p

- $K=E/(1-E)$, $W=KT_0(W,p)$ => Extragere W ca functie de p

Functia este numita functie de izoeficienta

Functia de izoeficienta pt. adunarea nr.

- Functia surplus pentru adunarea a *n* numere pe *p* elemente de procesare este aprox. $2p \log p$.
- Substituind *To* cu $2p \log p$ se obtine $W = K\, 2p \log p$.
- Functia de izoeficienta a acestui alg. este $O(p \log p)$.
  - Daca nr. de PEuri creste de la *p la p*', dimensiunea problemei *n* trebuie sa creasca cu un factor de $(p' \log p')/(p \log p)$ pentru a mentine o acceasi *E* cu *p* elemente de procesare.
- Remarca:
  - Surplusul datorat comunicatiilor este numai de *p* in acest caz.
  - In general, surplusul de comunicare depinde atat de dimensiunea problemei cat si de numarul de PEuri

Timpul minim de executie pentru + n nr.

- Timpul paralel pentru problema adunarii a *n* nr. pe *p* PEuri este $T_P = n/p + 2 \log p$.
- d $T_P/dp=0 \Rightarrow p=n/2$ obtinem $T_P^{min} = 2 \log p$.
- Timpul de executie cost-optimal minim pentru adunarea a n numere:
  - Timpul minim in care o problema poate fi rezolvata de un sistem paralel cosy-optimal
  - Dupa mai multe calcule (vezi textbook): $T_P^{cost\_opt} = 2 \log n - \log \log n$.

Alte masuri ale scalabilitatii

- Adecvate diferitelor cerinte ale sistemelor.
  - De ex. In aplicatii in timp real, obiectivul de a scala pentru a indeplini o sarcina pana la termen in timp:
    - Decompresie multimedia, cazul MPEG stream care trebuie decompresat la o rata de 25 cadre/seconda.
  - In numeroase aplicatii, dimensiunea maxima a problemei este constransa nu numai de timp, eficienta, ci si de memoria disponibila pe masina.
    - Metricile fac presupuneri relative la functia de crestere a memorie disponibile (cu nr. de PEuri)

Accelerarea scalata

- Este definita ca accelerarea obtinuta cand dimensiunea problemei creste liniar cu nr. de PEuri .
- Daca curba corespunzatoare este apropriata de o linie relativ la numarul de PEuri, atunci sistemul este considerat scalabil
- Metoda 1:
  - Dimensiunea problemei este crescuta pana la umplerea memoriei disponibile
  - Presupunere: memoria agregata creste cu numarul de PEuri.
- Metoda 2:
  - Dimensiunea problemei creste cu *p* limitandu-se la o anumita valoare a timpului de executie.

Scalarea constransa de memorie/timp

- Multiplicarea unei matrice n x n cu un vector:
  - $T_S = t_c n^2$, unde t_c este timpul pentru o singura operatie multiplicare-adunare, $T_P = t_c n^2/p + t_s \log p + t_w n$, $S = t_c n^2 / (t_c n^2/p + t_s \log p + t_w n)$.
  - Cerinta in termeni de memorie totala este de $\Theta(n^2)$.
  - Sa consideram.
    - Metoda 1:
      - presupunem ca memoria sistemului paralel creste linear cu numarul de PEuri, i.e., $m = \Theta(p) \Rightarrow n^2 = c \times p$, pentru anumita constanta c.
      - Accelarea scalata S' este: $S' = t_c c \times p / (t_c c \times p /p + t_s \log p + t_w \text{ sqrt}(c \times p))$ sau $S' = c_1 p/(c_2 + c_3 \log p + c_4 \text{ sqrt}(p))$.
    - Metoda 2: $T_P = O(n^2/p)$ si se contrange ca sa fie constanta, deci $n^2 = O(p)$ => acest caz este identic cu Metoda 2.
- Multiplicarea a doua matrici– vezi textbook
  - Metoda 1: S'= O(p).
  - Metoda 2: S"= O($p^{5/6}$)

Fractia seriala

- Fracția serială *f* determinată experimental poate fi utilizată pentru a cuantifica performanțele unui sistem paralel pe o problemă cu dimensiuni fixe.
- Fie cazul în care timpul de rulare în serie a unui calcul poate fi împărțit într-o componentă total paralelă și într-o componentă total serial, adică $W = Tser + Tpar$
- Ideal: $TP = Tser + Tpar / p$.
- Toate celelalte surpusuri, cum ar fi excesul de calcul și comunicare, sunt capturate în componenta seriala Tser.
- Fracția seriala f a unui program paralel este definită ca: $f = Tser/W$.
- $T_P = Tser + (W-Tser.)/p \Rightarrow T_P/W = f+(1-f)/p;$
- $S = W/T_P \Rightarrow 1/S = f+(1-f)/p \Rightarrow f=(1/S-1/p)/(1-1/p)$.
- Valorile mai mici ale f sunt mai bune, deoarece acestea conduc la eficiențe mai mari.
- Dacă f crește cu nr. PE-urile, atunci este considerat ca un indicator al creșterii comunicării aeriene și deci un indicator al scalabilității slabe.
- ***Exemplu:*** componentă în serie a produsului matrice-vector $f = (t_s\, p \log p + t_w n\, p)/ [t_c n^2 (p-1)]$ - numitor al acestei ecuații este timpul de execuție serial al alg.-ului. iar numărătorul corespunde surplusului în execuție paralelă.

Blocaje ale procesarii paralele

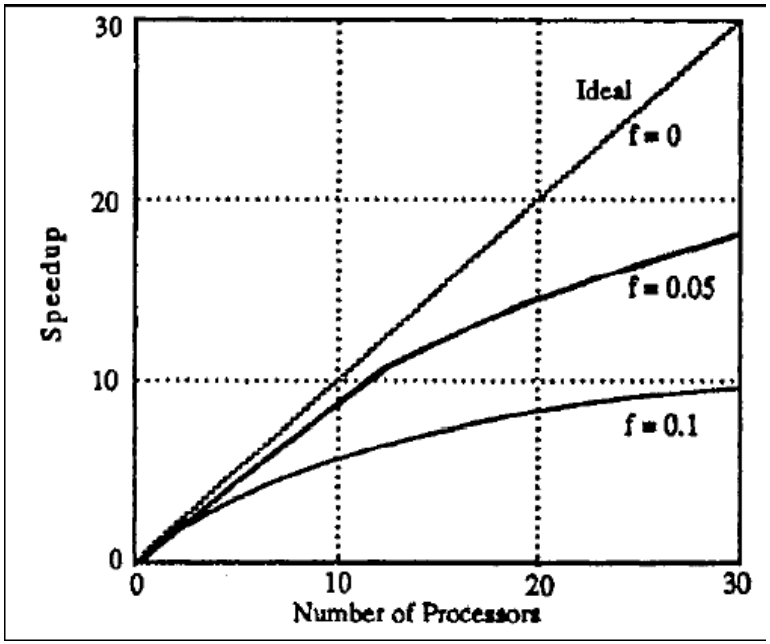
Legea lui Amdahl (1967)

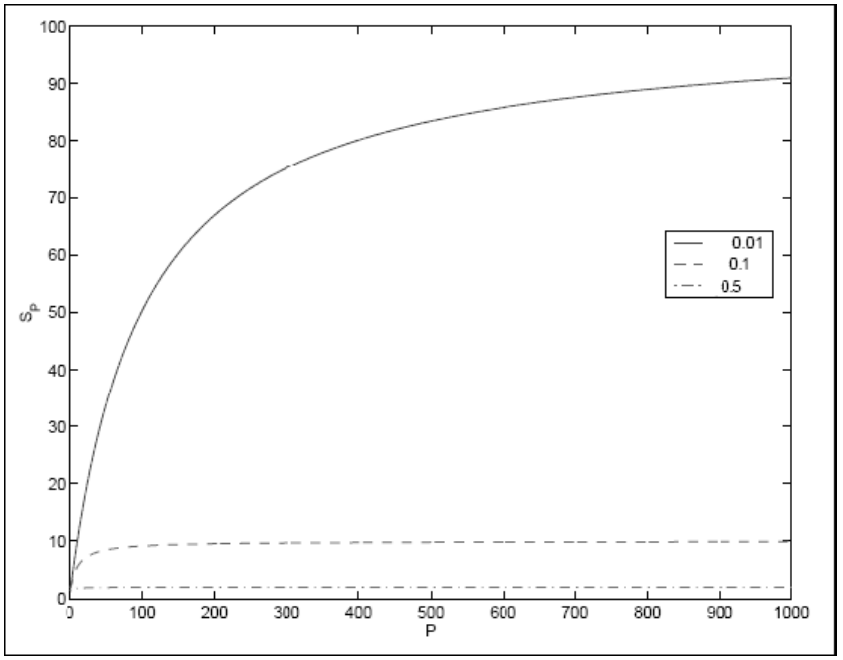
- A stabilit modul în care părțile mai lente ale unui algoritm influențează performanțele sale generale
  - întrucât părțile secvențiale ale unui algoritm sunt cele mai „lente”, legea Amdahl prevede că aceste părți au cel mai grav impact negativ asupra performanței generale.
- Afirmă că fracția *f* din calcul inerent secvențial limitează sever viteza care poate fi obținută cu procesoarele *p*
- Presupunem: o fracțiune 1-*f* din algoritm poate fi împărțită în părți p și ideal paralelizată, f de operațiunile rămase nu pot fi paralizate și astfel trebuie executate pe un singur procesor.

  $S = p/(fp + (1 - f))$ (slide-ul anterior).

  $f < 1 \Rightarrow Sp < 1/f.$
- *Accelerarea realizabilă prin calcul paralel este legată de valoarea care este invers proporțională cu fracția codului care trebuie executată secvențial*

Effectele legii lui Amdahl





Dacă f = 0,1, astfel încât 90% dintr-un algoritm poate fi paralelizat în mod ideal, iar dacă p = 10, S <6.

Dacă f = 0,01, ceea ce înseamnă că doar 1% din program nu este paralizabil, pentru p = 100 avem S = 50, deci acționăm la jumătate din eficiența maximă.

Comentarii

- derivarea legii lui Amdahl se bazează pe presupunerea că *f* este independentă de dimensiunea mărimii problemei *n*.
  - În practică, s-a observat că *f* scade în funcție de dimensiunea problemei.
  - Prin urmare, limita superioară a factorului de accelerare *S* crește, de obicei, în funcție de dimensiunea problemei.
- O altă anomalie este așa-numita accelerare superliniară, ceea ce înseamnă că *S*> *p*.
  - Acest lucru se poate întâmpla din cauza accesului la memorie și a unei gestionări greșite a memoriei cache sau din cauza faptului că implementarea seriala pe un PE este suboptimală.
- Există aplicații pentru care surplusul secvențial este foarte mic.
- Dacă calculul serial inițial este limitat de alte resurse decât disponibilitatea ciclurilor procesorului, performanța reală ar putea fi mult mai bună
  - O mașină paralelă mare poate permite reținerea unor probleme mai mari în memorie, reducând astfel paginarea memoriei virtuale,
  - Mai multe procesoare, fiecare cu propriul cache, pot permite să retina mult mai mult din problemă în cache.
- Legea presupune că pentru orice intrare dată, implementările paralele și seriale realizează exact același nr. a etapelor de calcul.
  - Dacă algoritmul serial utilizat în formulă nu este cel mai bun algoritm posibil pentru problemă, atunci un algoritm paralel inteligent care structurează diferit calculul poate reduce numărul total de pași de calcul

Legea lui Gustafson

- În loc să se întrebe cât de rapid ar rula un program serial dat pe o mașină paralelă, se întreabă cât timp ar fi durat un anumit program paralel pentru a rula pe un procesor serial.

  $T_{total}(1) = T_{setup} + pT_{compute}(p) + T_{finalization}$.

  Fractia serial scalata: $\gamma_{scaled} = (T_{setup} + T_{finalization})/T_{total}(p)$

  Astfel $T_{total}(1) = \gamma_{scaled}\ T_{total}(p) + p(1 - \gamma_{scaled})\ T_{total}(p)$.

- Rescrierea ecuației pentru accelerare și simplificare:

  viteză la scară (sau la timp fix) $S(P) = P + (1-P)\ \gamma_{scaled}$.

  (Ecuatia lui Gustafson)

- Luam limita in *p* tinand $T_{compute}$ si astfel γ_{scaled} constante.
  - Interpretarea este că creștem dimensiunea problemei, astfel încât timpul de rulare total să rămână constant atunci când sunt adăugate mai multe procesoare.
  - Aceasta conține presupunerea implicită că timpul de execuție al termenilor seriali nu se modifică pe măsură ce dimensiunea problemei crește.

=> In acest caz, acceleratia este liniara in P *!* => dacă problema crește pe măsură ce se adaugă mai mulți PEuri, legea lui Amdahl va fi pesimistă!

Other laws

1. ***Legea lui Grosch:*** puterea de calcul este proporțională cu pătratul de cost
   - Legea lui Grosch a fost formulată în zilele de inceput a calculului paralel și a fost valabilă pentru primele mașini.

2. ***Conjunctura lui Minsky':*** *S* este proportional cu logaritm din *p*
   - Accelerarea reală poate varia de la log *p* la *p* (*p* / log *p* fiind o valoare medie acceptabila).

3. ***Inertia software:*** costul pentru convertirea programelor in variant paralela si retinerea programatorilor cu asemenea cunostiinte este prohibit
   - Studenții sunt instruiți să gândească paralel.
   - Instrumentele sunt dezvoltate pentru a transforma automat codul secvențial în cod paralel.

Analiza asimptotică a programelor paralele

Evaluarea unui set de programe paralele pentru rezolvarea unei probleme date

- Exemplu: sortare
- Cele mai rapide programe seriale pentru această problemă rulează în timp O (*n* log *n*).
- Să analizăm patru algoritmi paraleli diferiți A1, A2, A3 și A4, pentru a sorta o listă dată.
- Obiectivul acestui exercițiu este de a determina care dintre acești patru algoritmi este cel mai bun

	A1	A2	A3	A4
p	n^2	log *n*	*n*	O(*n*)
T_P	1	*n*	O(*n*)	O(*n* log *n*)
S	*n* log *n*	log *n*	O(*n* log *n*)	O(*n*)
E	Log n /n	1	Log n/ O(n)	1
pT_P	n^2	*n* log *n*	$n^{1.5}$	*n* log *n*

Exemplu de sortare

- Cea mai simplă măsură este cea a vitezei
  - algoritmul cu cel mai mic T_P este cel mai bun.
  - prin această măsură, algoritmul A1 este cel mai bun, urmat de A3, A4 și A2
- Utilizarea resurselor este un aspect important al proiectării programelor
  - Rar avem n^2 PE-uri cum sunt cerute de algoritm A1.
  - Această valoare a evaluării algoritmului prezintă o imagine cu totul diferită: alg. A2 și A4 sunt cei mai buni, urmati de A3 și A1.
- Cost:
  - Ultimul rând al tabelului prezintă costul celor patru algoritmi.
  - Costurile algoritmilor A1 și A3 sunt mai mari decât timpul de rulare serial $n \log n$ și, prin urmare, niciunul dintre acești algoritmi nu este optim.
  - Algoritmii A2 si A4 sunt cost-optimali.
- Concluzie:
  - Este important să se înțeleagă mai întâi obiectivele analizei algoritmului paralel și să se utilizeze valori adecvate, deoarece utilizarea diferitelor valori poate duce adesea la rezultate contradictorii.