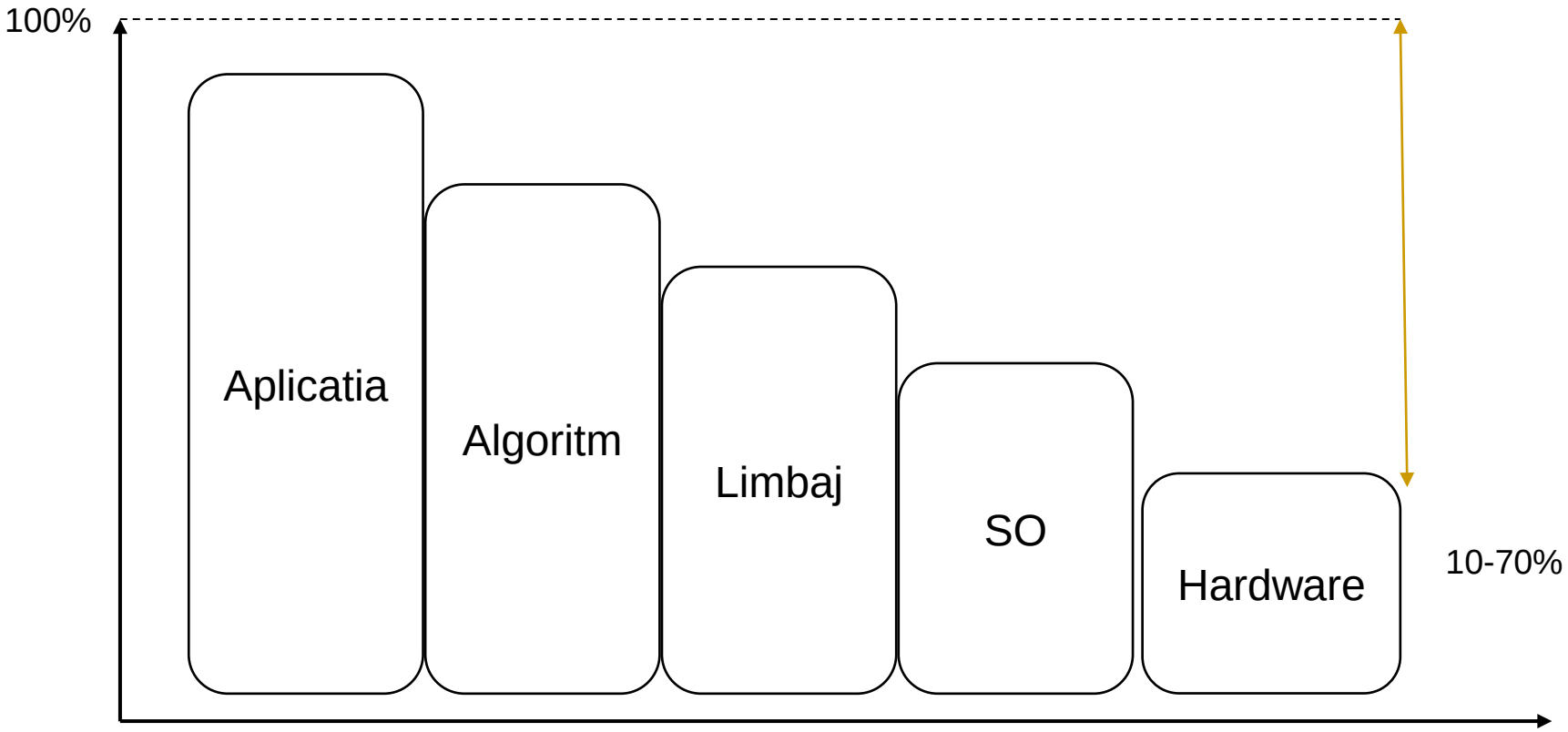
# Problematici asociate calculului paralel

# Aspecte ale prelucrarii paralele

- Problemele de baza in calculul parallel se refera la urmatoarele aspecte:
- Stabilirea granularitatii task-urilor
- Elaborarea algoritmilor cu paralelism intrinsec
- Proiectarea limbajelor de programare
- Proiectarea unor arhitecturi hardware adecvate
- Proiectarea unor sisteme de operare adecvate

# Grad de paralelism



# Nivelurile de paralelism

- Paralelismul poate fi examinat la multe niveluri in functie de complexitatea dorita. Frecvent gasim descrierea paralelismului la nivelurile:

* job $JOB = \{ JOB_1, \ldots, JOB_i, \ldots\ldots, JOB_m \}$
* task $JOB_i = \{ Ti_1, \ldots, Ti_j, \ldots..., Ti_n \}$
* proces $Ti_j = \{Pi_{j1}, \ldots., Pi_{jk}, ..., Pi_{jp}\}$
* thread $Pi_{jk} = \{THi_{jk1} \ldots THi_{jkt}\}$
* variabila
* instructiune
* bit

# Nivelurile de paralelism

***Nivel de job-uri***

Se executa doua sau mai multe programe independente pe resurse de procesare distincte.

- $JOB = \{ JOB_1, \ldots, JOB_i, \ldots\ldots, JOB_m \}$

***Nivel de task-uri***

- $JOB_i = \{ Ti_1, \ldots, Ti_j, \ldots\ldots, Ti_n \}$

Fiecare Ti se executa pe cate un procesor distinct insa exista o relatie intre Ti si Tj in ceea ce priveste transferul de date sau completarea functiilor de prelucrare realizate in cadrul job-ului.

Exemplu:

Consideram un robot care are mai multe grade de libertate. Pentru fiecare grad de libertate exista un procesor. Programul de conducere a robotului este partitionat in task-uri care se ocupa de cite un grad de libertate al robotului.

Taskurile se executa in paralel, dar interactioneaza pentru miscarea robotului.

# Nivel de proces

* $Ti_j = \{Pi_{j1}, ….,Pi_{jk}, ..., Pi_{jp}\}$

Task-urile sunt alcatuite din mai multe procese care in general sunt identificate de utilizator sau de catre compilator daca acesta este destinat pentru structuri multiprocesor.
Sa consideram task-ul

```
for i=1 to n
        x(i)=x(i-2)+y(i-2)
        y(i)=x(i-2) * y(i-1)
        suma(i)=x(i) +y(i)
        if s(i) > a then c=c+1
                    else d=d+1
 end for
```

In cadrul acestui task identificam doua procese:

```
P1(i)
        x(i)=x(i-2) + y(i-2)
        y(i)=x(i-2) * y(i-1)
P2(i)
        suma(i)=x(i) +y(i)
        if s(i) > a then c=c+1
                    else d=d+1
```

Intre P1(i) si P2(i) exista o dependenta de date, deci nu pot fi efectuate in paralel
Insa P1(i) si P2(i-1) pot fi efectuate in paralel

# Nivel de proces

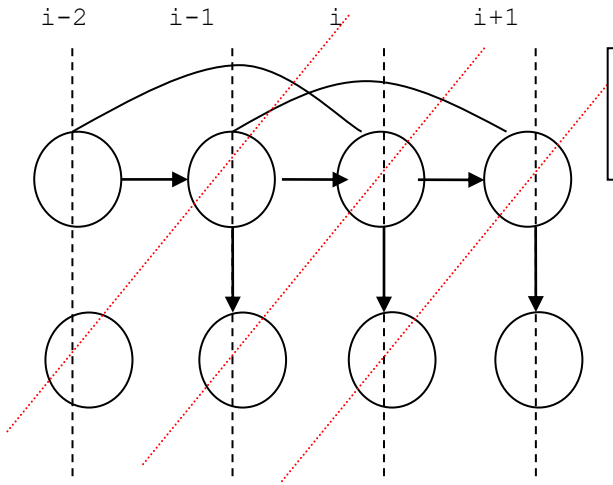
- Interdependenta intre aceste procese este urmatoarea

P1
x(i)=x(i-2)+y(i-2)

y(i)=x(i-2) * y(i-1)

P2
suma(i)=x(i) +y(i)

if sum a(i) > a then c=c+1
else d=d+1




Se pot executa simultan:
P1(i) si P2(i-1)

# Nivel de variabila

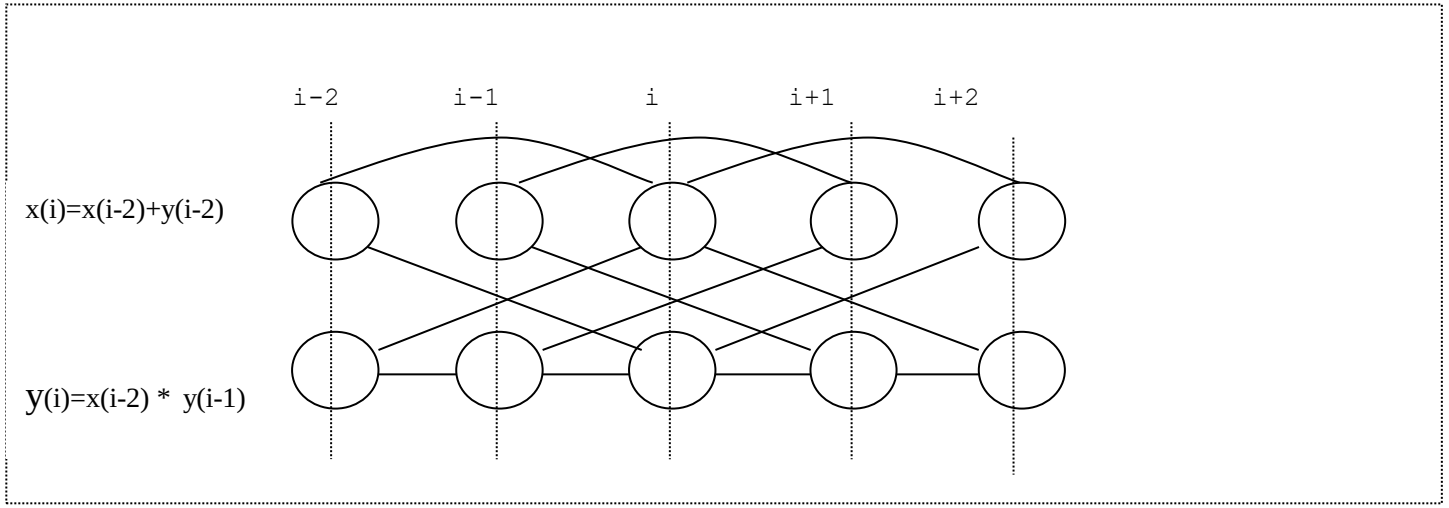
Pi= { Ii1, ….Iik}

Fiecare proces este format dintr-uo multime de instructiuni care pot calcula variabilele de iesire in functie de variabilele de intrare.

Paralelismul in cadrul unui proces se poate realiza prin calculul simultan a mai multe variabile de iesire.

P1
x(i)=x(i-2)+y(i-2)
y(i)=x(i-2) * y(i-1)

In exemplul anterior putem considera ca in cadrul procesului P1 variabilele x(i) si y(i) se pot calcula in paralel
sau
x(i) si y(i+1) se pot calcula in paralel



# Nivel de Instructiune

Generarea codului, pentru calculul expresiei z =x + y, in structura pipeline

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| lw r1, x | IF | ID | EX | MEM | WB | | | | |
| lw r2, y | | IF | ID | EX | MEM | WB | | | |
| add r3, r1, r2 | | | IF | ID | stall | EX | MEM | WB | |
| sw z, r3 | | | | IF | stall | ID | EX | MEM | WB |

# Exemplu

- Să luăm în considerare un procesor care implementează o structura paralela pipeline de citire-interpretare-executie pentru procesare suprascalară.
- Arătați îmbunătățirea performanței față de procesarea scalară cu pipeline și procesarea fără pipeline, presupunând un ciclu de instrucțiuni contine:
  - • citire care necesita o singura perioada de ceas
  - • decodarea instructiunii necesita  două perioade de ceas
  - • executia instructiunii necesita trei perioade de ceas
- și avem o secvență de 200 de instrucțiuni:

# Exemplu solutie

- O structure fara pipeline necesita 1200 de cicluri de ceas: (nx6)

200 * (1 + 2 + 3) = 1200

n * (1 + 2 + 3)

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 |

- O structura pipeline scalară necesita 603 cicluri de ceas: ((n-1) * 3 +6)

1 + 2 + (200 * 3) = 603

1 + 2 + (n * 3)

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | | | | | | | | | | |
| | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | | | | | | | |
| | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | | | | |
| | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | |
| | | | | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | |
| | | | | | | | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | |

# pipeline superscalară cu două unități

- O structura pipeline superscalară cu două unități paralele ar necesita 303 cicluri de ceas (n / 2 + 1) * 3)

1 + 2 + ((200/2) * 3) = 303

1 + 2 + ((n / 2) * 3)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D1 | D2 | E1 | E2 | E3 | | | | | | |
| F | D1 | D2 | E1 | E2 | E3 | | | | | | |
| | | F | D1 | D2 | E1 | E2 | E3 | | | | |
| | | F | D1 | D2 | E1 | E2 | E3 | | | | |
| | | | | F | D1 | D2 | E1 | E2 | E3 | | |
| | | | | F | D1 | D2 | E1 | E2 | E3 | | |
| | | | | | | F | D1 | D2 | E1 | E2 | E3 |
| | | | | | | F | D1 | D2 | E1 | E2 | E3 |

Sau daca fetch-ul se realizeaza secvential

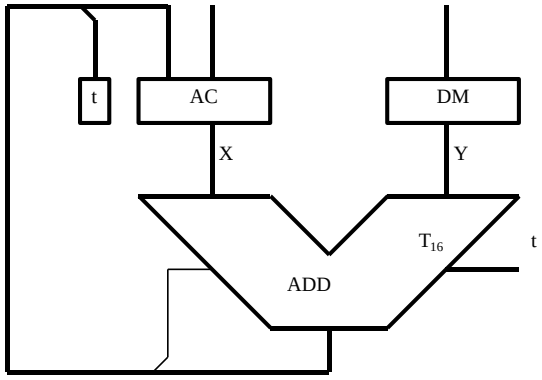
1+2+((200/2)*4= 403

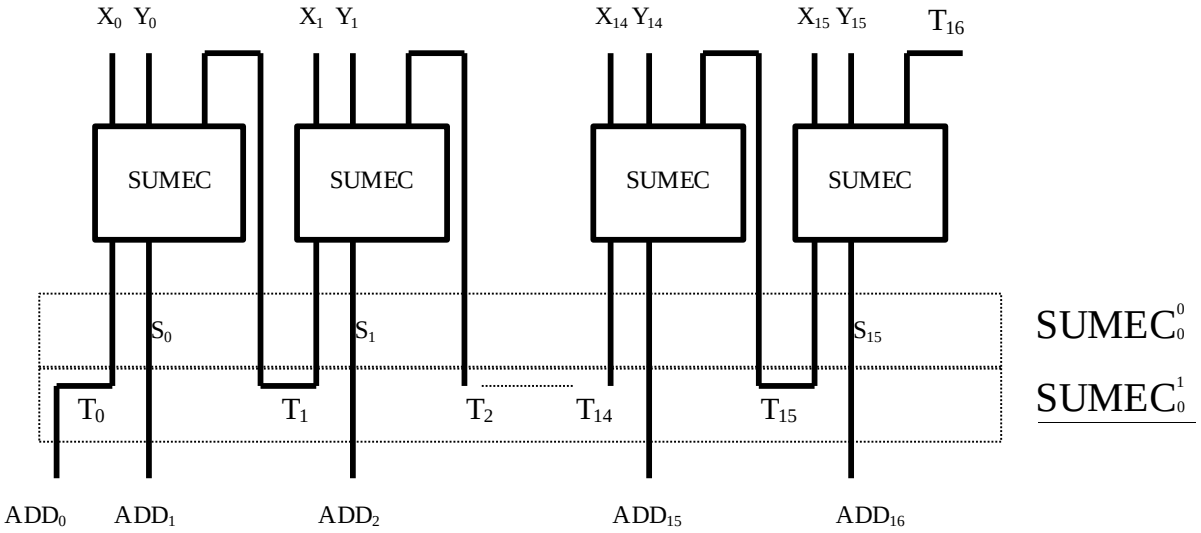
1 + 2 + ((n / 2) * 4)

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D1 | D2 | E1 | E2 | E3 | | | | | | | | | |
| | F | D1 | D2 | E1 | E2 | E3 | | | | | | | | |
| | | | F | D1 | D2 | E1 | E2 | E3 | | | | | | |
| | | | | F | D1 | D2 | E1 | E2 | E3 | | | | | |
| | | | | | | F | D1 | D2 | E1 | E2 | E3 | | | |
| | | | | | | | F | D1 | D2 | E1 | E2 | E3 | | |
| | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 |
| | | | | | | | | | | F | D1 | D2 | E1 | E2 | E3 |

# Nivel bit

Toate calculatoarele, cu foarte putine exceptii, utilizeaza unitati aritmetice paralele cu sau fara anticiparea transportului si cu structura pipeline.



# Ce este performanța?

- În calcul, performanța este definită de 2 factori
  - Cerințe de calcul (ce trebuie făcut)
  - Resurse de calcul (cat costă să o faci)
- Problemele de calcul se traduc în cerințe
- Resurse de calcul interacțiune și compromise
- Performanța în sine este o măsură a cât de bine cerințele de calcul pot fi satisfăcute
- Evaluăm performanța pentru a înțelege relațiile dintre cerințe și resurse
- Decideți cum să schimbați „soluțiile” pentru a atinge obiectivele
- Măsurile de performanță reflectă deciziile despre cum și cât de bine „soluțiile” sunt capabile să satisfacă cerințe de calcul

# problemele de performanță

Aici ne preocupă problemele de performanță când folosind un mediu de calcul paralel

- Performanță în raport cu calculul paralel
- Performanța este rațiunea de a fi pentru paralelism

Performanță paralelă versus performanță secvențială

Dacă „performanța” nu este mai bună, paralelismul nu este necesar

Prelucrarea în paralel include tehnici și tehnologii necesare pentru a calcula în paralel

- Hardware, rețele, sisteme de operare, biblioteci paralele,
- Limbajele de proramare, compilatoare, algoritmi, instrumente, ...

Paralelismul trebuie să ofere performanță

Cum? Cât de bine?

# Așteptarea performanței

- Dacă fiecare procesor este evaluat la k  MFLOPS și există p procesoare, ar trebui să vedem performanța k * p  MFLOPS?
  - Dacă durează 100 de secunde la un procesor, nu ar trebui sa dureze 10 secunde pe 10 procesoare?
- Mai multi factori  afectează performanța
- Fiecare trebuie înțelesi separat
- Dar ei interacționează între ei în moduri complexe
  - Soluția la o problemă poate crea alta
  - O problemă poate masca alta problema, etc.
- Scalarea (sistem, dimensiunea problemei) poate schimba condițiile
- Trebuie să înțelegeți limitele de performanță

# Calcule paralele

- Un calcul paralel “jenant” este unul care poate fi evident împărțit în task-uri independete complet care pot fi executate simultan
  - Într-un calcul paralel cu **adevărat jenant**, nu există interacțiunea între procese separate
  - Într-un calcul paralel **aproape jenant** rezultă calcule ce trebuie distribuite și colectate / combinate într-un fel
- Calculele paralel jenante au potențial pentru a atinge viteza maximă pe platforme paralele
  - Dacă este nevoie de timp T secvențial, există potențialul de a realiza timpul T / P care rulează în paralel cu P procesoare
  - De ce acest lucru nu este întotdeauna adevarat?

# Relatia intre algoritmi paraleli si arhitecturi paralele

- Prelucrarea paralela include atat algoritmi paraleli cat si arhitecturi paralele.
- Un algoritm paralel poate fi considerat ca o colectie de procese independente care se executa simultan, procesele comunicand in timpul executiei.
- Astfel un algoritm se executa pe unitati functionale hardware care in general constau din
  - elemente de procesare;
  - module de transfer date.

- Ceea ce intereseaza este cum se transpun procesele pe unitatile functionale hardware.

# Algoritmi paraleli / Arhitecturi paralele

- H.T.Kung a fost unul din primii care au studiat relatia intre algoritmi si arhitectura. A stabilit citeva caracteristici si a prezentat corelarea intre ele:

| Algoritmi paraleli | Arhitecturi paralele |
|---|---|
| granularitate modul | complexitate procesor |
| control concurent | mod de operare |
| mecanismul datelor | structura memoriei |
| geometria comunicatiei | retele de comutare |
| .complexitate algoritm | numar de procesoare; dimensiune memorie |

# Algoritmi - Arhitectura

- **Granularitate modul (obiect)** - se refera la complexitatea modulului care poate fi job, task, proces sau instructiune.
  - De obicei exista posibilitati de paralelism la o granularitae mare dar care nu este exploatat deoarece implica o comunicatie intensa si o crestere a complexitatii software-ului.
  - La acest nivel se face o analiza intre granularitate si comunicatie pentru a stabili solutia cea mai buna.
- **Control concurent** se refera la schema de selectie a modulelor pentru executie.
  - Aceasta trebuie sa satisfaca dependenta de date si dependenta de control (asigurarea excluderii mutuale a accesului la aceeasi resursa).

  Citeva scheme de control sunt bazate pe:
  - disponibilitatea datelor (data flow)
  - control centralizat (sinchronized)
  - cereri (demand-driven).
- exemplu algoritmii care prelucreaza matrice se potrivesc foarte bine pe procesoarele sistolice sau pe masive de procesoare iar algoritmii care au transferuri conditionate si alte iregularitati se potrivesc foarte bine pe arhitecturi asincrone cum ar fi multiprocesoare si data-flow.

- **Mecanismul datelor** se refera la faptul cum sunt utilizati operanzii.
  - Datele furnizate de instructiuni pot fi utilizate ca "date pure" in structurile data-flow sau pot fi depuse in locatii adresabile in masinile Von Neumann.
- **Geometria comunicatiei** - se refera la sablonul de interactiune intre module.
  - Geometria comunicatiei poate fi regulata sau neregulata.
- **Complexitate algoritm** se refera la numarul de operatii necesare pentru implementarea algoritmului.
  - Are influenta asupra numarului de procesoare si asupra dimensiunii memoriei.

# Performanță și scalabilitate

- Evaluare
  - Runtime secvențial (Tseq) este o funcție de
    - dimensiunea problemei și arhitectura
  - Runtime paralel (Tpar) este o funcție de
    - dimensiunea problemei și arhitectura paralelă
    - numarul de procesoare utilizate în execuție
- Performanță paralelă afectată de
  - algoritm + arhitectură
- Scalabilitate
  - Abilitatea algoritmului paralel de a atinge performanța câștigă proporțional cu numărul de procesoare și cu dimensiunea problemei

# Scalabilitate

- Un program se poate utiliza mai multe procesoare
  - Cum evaluezi scalabilitatea?
- Cum evaluezi performantele scalabilității?
- Evaluare comparativă
  - Dacă se dublează numărul de procesoare, la ce să ne așteptăm?
  - Scalabilitatea este liniară?
- Utilizați o măsură de eficiență paralelă
  - Este menținută eficiența pe măsură ce crește dimensiunea problemei?
- Aplicați valori de performanță

# Indicatori de performantele ai calculului paralel

Datorita complexitatii calculului paralel este greu de a stabili o masura a performantelor care sa stabileasca real si absolut performantele unui sistem cu arhitectura paralela.
Totusi sunt utilizati cativa indicatori care masoara diferite aspecte globale.

***Rata de executie***

masoara rata de producere a unor rezultate in unitatea de timp.

Uzual se folosesc :

- **MIPS / GIPS / TIPS** – milioane /giga/tera de instructiuni pe secunda care masoara numarul de instructiuni pe care le executa pe secunda o unitate de prelucrare mono sau multiprocesor. Un astfel de indicator este inadecvat pentru o masina SIMD care executa aceeasi instructiune pe mai multe fluxuri de date.
- **MOPS** / **GOPS / TOPS** - milioane /giga/tera de operatii pe secunda care masoara operatiile efectuate de unitatile de prelucrare. O astfel de unitate de masura nu tine seama de lungimea cuvintului si nici de natura operatiilor.
- **MFLOPS** / **GFLOPS / TFLOPS / PFLOPS-** milioane /giga/tera/peta de operatii cu virgula mobila pe secunda (Giga) care masoara operatiile cu virgula mobila efectuate (Tera) de unitatile de prelucrare.O astfel de masura este adecvata numai pentru aplicatii numerice dar nu reprezinta nici un fel de masura pentru prelucrari alfanumerice sau pentru aplicatii bazate pe inteligenta artificiala.
- **MLIPS** / **GLIPS/ TLIPS**– milioane /giga/tera de inferente logice pe secunda care masoara numarul de inferente logice realizate in aplicatii de IA

# Indicatori

*Presupunem ca avem o structura cu p procesoare, care participa la rezolvarea unei probleme.*

***Viteza de prelucrare - $V_p$***

$$V_p = \frac{T_1}{T_p}$$ unde

$T_1$ - timpul necesar pentru prelucrare utilizind un singur procesor

$T_p$ - timpul necesar pentru prelucrare utilizind **p** procesoare

Cu alte cuvinte, $V_p$ reprezinta raportul intre prelucrarea secventiala si cea paralela care scoate in evidenta cresterea in viteza datorita paralelismului.

deoarece se consuma timp cu sincronizarea, comunicarea si "overhead" cerut de interactiunea intre procesoare.

$1 \leq V_p < p$

***Eficienta $E_p$***

Este definita ca raportul intre viteza de prelucrare si numarul de procesoare.

$$E_p = \frac{V_p}{p} = \frac{T_1}{p * Tp} < 1$$ Eficienta este o masura a eficientei costului

Cost $p * T_p$ este $p * T_p >> T_1$

# Indicatori

### *Redundanta Rp*

Este definita ca raportul intre numarul total al operatiilor Op necesar efectuarii calculului cu p procesoare si numarul de operatii O1 necesar efectuarii calculului cu un singur procesor.

$$R_p = \frac{O_p}{O_1} = \frac{\textbf{nt total de operatii efectuate pe cele p procesoare}}{\textbf{nr de operatii necesare efectuarii pe 1 procesor}}$$

*reflecta timpul pierdut cu overhead-ul.*

### *Utilizarea Up*

Este definita ca raportul dintre numarul de operatii $O_p$ necesar efectuarii calculului cu p procesoare si numarul de operatii care ar fi putut fi efectuate cu p procesoare in timpul $T_p$

$$U_p = \frac{O_p}{p * Tp} \leq 1$$

# Limite ale calculului paralel

Este foarte important a stabili limita calculului paralel.
Fie Tn timpul necesar executiei a n taskuri de k tipuri diferite.
Fiecare tip consta din ni taskuri necesitind ti secunde fiecare

$$Tn = \sum_{i=1}^{k} (ni*ti) \qquad \text{iar} \qquad n = \sum_{i=1}^{k} ni$$

Tn timpul necesar executarii celor n task-uri

Prin definitie, rata de executie R este numarul de operatii efectuate in unitatea de timp

$$Rn = \frac{n}{Tn} = \frac{n}{\sum_{i=1}^{k} (ni*ti)}$$

Fie $fi=ni/n$ $\quad \sum_{i=1}^{k} fi = 1 \quad$ $Ri = \frac{1}{ti}$

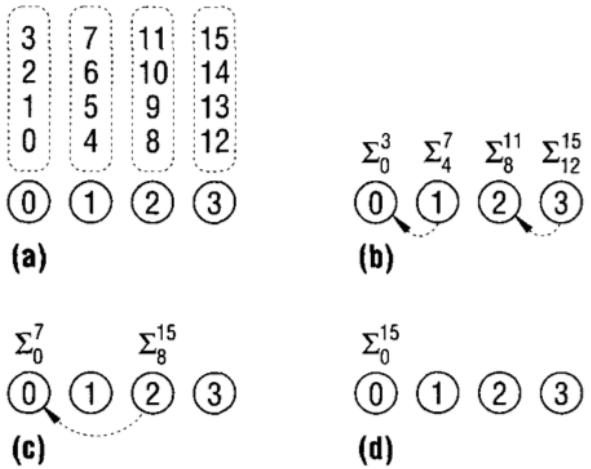
In acest caz rezulta

$$Rn = \frac{n}{\sum_{i=1}^{k} (ni*ti)} = \frac{1}{\sum_{i=1}^{k} (ni/n*ti)} = \frac{\mathbf{1}}{\mathbf{\sum_{i=1}^{k} (fi/Ri)}}$$

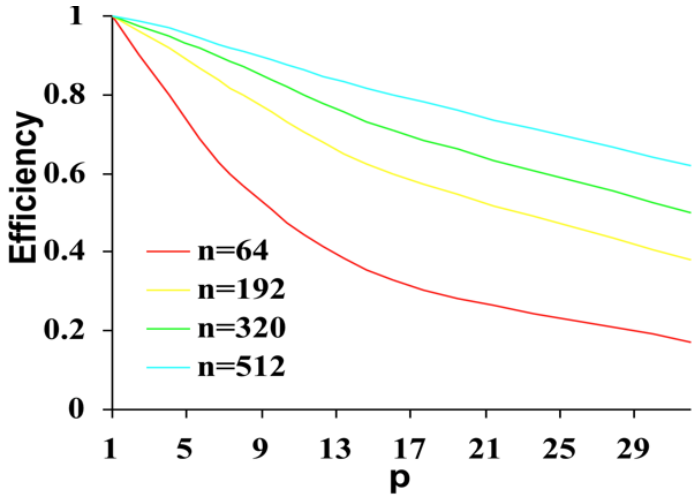
Aceasta relatie reprezinta "bottleneck" - limitarea in structurile paralele.

# Scalabilitatea in a aduna n numere

- Scalabilitatea unui sistem paralel este masura capacitatii de a  creste viteza de prelucrare utilizand mai multe procesoare
- Adunarea a n numere utilizand p procesoare conduce la:




- $T_p = \frac{n}{p} + 2 * \log p$
- $V_p = \frac{n}{\frac{n}{p} + 2 * \log p}$
- $\boldsymbol{E_p = \frac{V_p}{p} = \frac{T_1}{p * Tp} = \frac{n}{n + 2 * p * \log p}}$



# Legea lui Amdahl

Sa consideram

*f = fseq procentul de program care se executa secvential*

*1-f = fpar procentul de program care poate fi paralelizat*

Fie $T_1$ *timpul de executie pe o structura cu 1 procesor*

Fie $T_p$ *timpul de executie pe o structura cu p procesoare*

*Vp* viteza de prelucrare

$Vp = T1 / Tp = = T1 / (f*T1 + (1-f)*T1 / p)) = 1 / (f + (1-f)/p))$

*Sau*

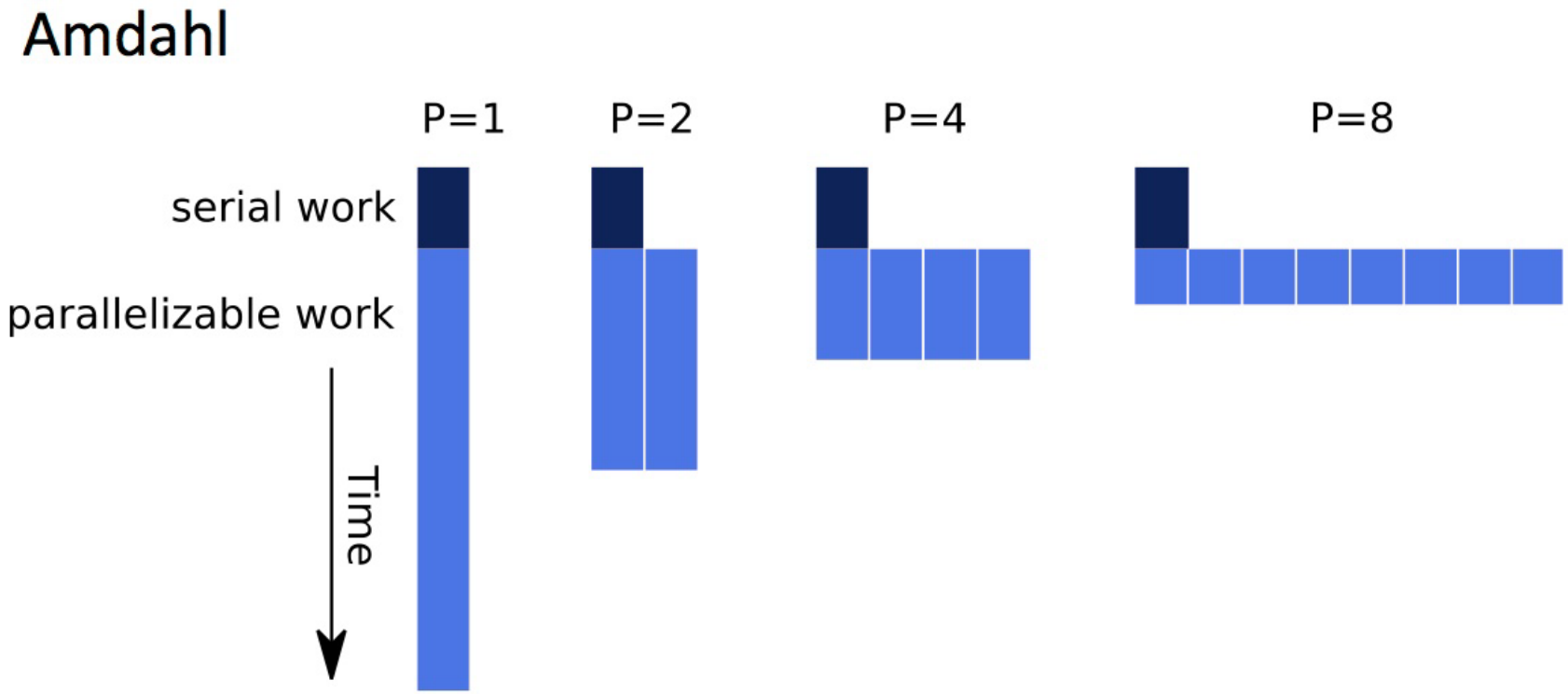
$Vp = 1 / (fseq + fpar/p)$

$Vp = 1 / (fseq + (1-fseq)/p)$

Cand $p \rightarrow \infty$

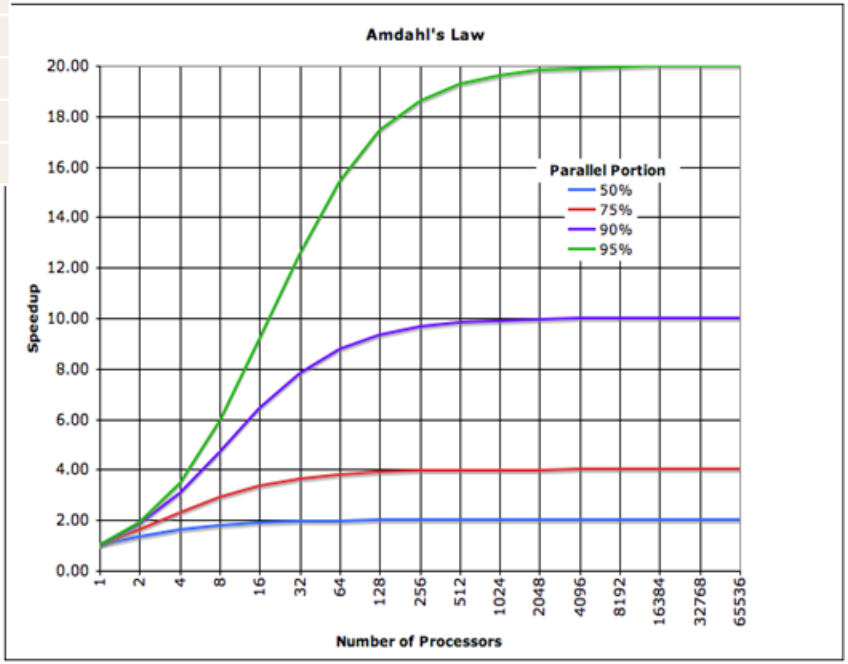
$Vp = 1 / f$ *sau* $Vp = 1 / fseq$

## *Amdahl*



# Cresterea in Viteza

| | | Crestere viteza | | |
|---|---|---|---|---|
| | | Paralelism | | |
| | *f=100%* | *f=50%* | *f=10%* | *f=1%* |
| Numar procesoare | *(1-f)=0%* | *(1-f)=50%* | *(1-f)=90%* | *(1-f)=99%* |
| 1 | 1 | 1.00 | 1.00 | 1.00 |
| 2 | 1 | 1.33 | 1.82 | 1.98 |
| 5 | 1 | 1.67 | 3.57 | 4.81 |
| 10 | 1 | 1.82 | 5.26 | 9.17 |
| 100 | 1 | 1.98 | 9.17 | 50.25 |
| 1,000 | 1 | 2.00 | 9.91 | 90.99 |
| 10,000 | 1 | 2.00 | 9.99 | 99.02 |
| 100,000 | 1 | 2.00 | 10.00 | 99.90 |
| 1,000,000 | 1 | 2.00 | 10.00 | 99.99 |
| 10,000,000 | 1 | 2.00 | 10.00 | 100.00 |



# Scalabilitate

- *Capacitatea algoritmului paralel de a obține câștiguri de performanță proporțional cu numărul de procesoare și dimensiunea problemei*

*Când se aplică Legea lui Amdahl?*

- *Când dimensiunea problemei este fixa*
- *Scalare puternică ($p \to \infty$, $Vp = V_\infty \to 1 / f$)*

*Limita de accelerare este determinată de gradul de secvențialitate, nu de numarul de procesoare !!!*

- *Eficiența perfectă este greu de realizat*

# Legea lui Gustafson-Barsis (accelerare scalată)

Un Data center este interesat , cand se pune problema scalarii, de probleme mai mari

- Cât de mare poate fi o problemă (HPC Linpack)
- Care este contrangerea problemei care se executa in paralel

Să presupunem că timpul de executie este menținut constant

- $Tp = C = (f + (1-f)) * C = (fseq + fpar) * C$
- $fseq$ este fracțiunea de $Tp$ cheltuită în execuția secvențială
- $fpar$ este fracțiunea de $Tp$ cheltuită în execuție paralelă

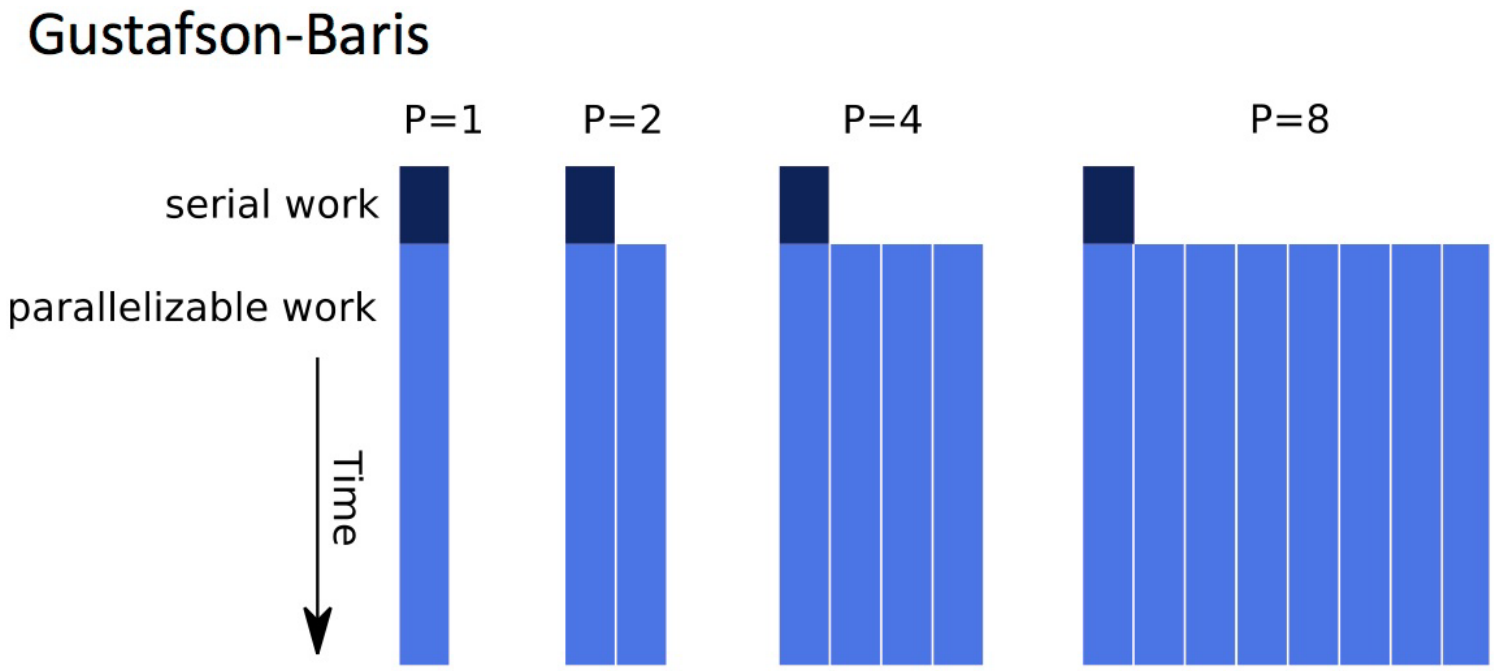
Scalara pe p procesoare conduce la

- $Ts = fseq + p* (1- fseq) = fseq + p* fpar$, *deoarece* $fpar = 1- fseq$
- $p*fpar$ *ne arata cat putem incarca structura din centrul de date*
- $Ts = 1- fpar + p*fpar = 1 + (p-1)fpar$
- $Tn = fseq + (1-fseq)*p / p = 1$

Care este viteza în acest caz?

$Vp = Ts / Tp = Ts / 1 = fseq + p(1 – fseq) = 1 + (p-1)\, fpar$

# Gustafson - Baris



# *Gustafson-Barsis' / Scalabilitate*

Scalabilitate

- Capacitatea algoritmului paralel de a atinge performanța proporțional cu numărul de procesoare și cu dimensiunea problemei

Când se aplică Legea lui Gustafson?

- Când dimensiunea problemei poate crește ca număr
- Scalare slabă $Vp = 1 + (p\text{-}1)fpar$
- Funcția de accelerare include numărul de procesoare !!!
- Poate menține sau crește eficiența paralelă prin scalare problema / probleme

# Sun and Ni

- Introduce modelul bazat pe limita de memoriei
- $Ts = fseq + g(p)(1- fseq)$
  - *unde G(n) este cresterea de memorie a unui procesor*
- $Tp = fseq + g(p)(1- fseq) /p$
- $Vp=Ts/Tp= (fseq + g(p)(1- fseq)) /(fseq + g(p)(1- fseq) /p)$

- *Caz 1 : g(n)=1*
- $Vp= (fseq + 1- fseq) /(fseq + (1- fseq) /p)= \mathbf{1 /(fseq + (1- fseq) /p)} \sim Amdahl$

- *Caz 2 : g(n)=p*
- $Vp=Ts/Tp= (fseq + p(1- fseq)) /(fseq + p(1- fseq) /p)=(fseq + p(1- fseq)) /1 =$ $\mathbf{fseq + p(1- fseq)} \sim Gustafson$

- *Caz 3: g(p)= m > p*
- $Vp=Ts/Tp= (fseq + m(1- fseq)) /(fseq + m(1- fseq) /p)=$
- $\mathbf{(fseq + m(1- fseq))/(m/p+ (1-m/p)fseq)}$

---

- *Workload-ul creste mai repede decat cererea de memorie.*

# Limita lui Worlton

Jack Worlton a studiat limitele calculului paralel utilizind un model care aproximeaza operarea unui multiprocesor.

El presupune ca un program paralel consta din :

- sectiuni de prelucrare distribuite pe diverse procesoare;
- sectiuni de sincronizare;
- sectiuni care se executa secvential si constituie un "overhead".

Fie

ts timpul de sincronizare;
to sectiune de overhead
t media timpului de executie a unui task
N numarul de task-uri (intre puncte de sincronizare)
P numarul de procesoare

**Timpul de executie** secvential al celor N task-uri este :

$T1 = N*t$

**Timpul de executie** a celor N task-uri executate pe p procesoare este :

$Tn,p = ts + (\lceil N / p \rceil)*(t+to)$

**Viteza de prelucrare**

$$Vn,p = T1 / Tn,p = \frac{N*t}{ts + (\lceil N / p \rceil)*(t+to)} = \frac{1}{ts / (N*t) + (1/N)*(\lceil N / p \rceil)*(1+(to/t))}$$

# Comentarii Worlton

Observatii:

Pentru a creste viteza de prelucrare trebuie sa avem in vedere reducerea efectului sincronizarii, overhead-ului si a numarul de pasi $\lceil N / p \rceil$

- ***efectul sincronizarii*** cauzat de ts/(N*t) poate fi redus fie prin micsorarea timpului de sincronizare ts sau prin marirea intervalului intre sincronizari N*t
- ***efectul overhead-ului*** cauzat de to/t poate fi redus prin reducerea timpului de overhead to sau prin cresterea granularitatii task-urilor.

*Cresterea granularitatii task-urilor ajuta la reducerea atat a efectului sincronizarii cat si a overhead-ului.*

- ***numarul pasilor de calcul*** $\lceil N / p \rceil$ poate fi redus prin cresterea numarului de procesoare si avand un numar de task-uri multiplu de procesoare.

$$V_{n,p} = T1 / T_{n,p} = \frac{N*t}{ts + (\lceil N / p \rceil)*(t+to)} = \frac{1}{ts / (N*t) + (1/N)*(\lceil N / p \rceil)*(1+(to/t))}$$

# Eficienta

$$V_{n,p} = T1 / T_{n,p} = \frac{N*t}{ts + (\lceil N / p \rceil)*(t+to)} = \frac{1}{ts / (N*t) + (1/N)*(\lceil N / p \rceil)*(1+(to/t))}$$

**Eficienta**

$$E_{n,p} = \frac{V_{n,p}}{p}$$

Presupunind ca avem un numar mare de task-uri si ca overhead-ul este neglijabil relativ la granularitate avem :

p >>>> N fix

$$V_{n,p} = \frac{N}{ts/t} \quad \text{si} \quad E_{n,p} = 0$$

N >>>> p fix

$$V_{n,p} = \frac{P}{1 + to/t} \quad \text{si} \quad E_{n,p} = \frac{1}{1 + to/t}$$