

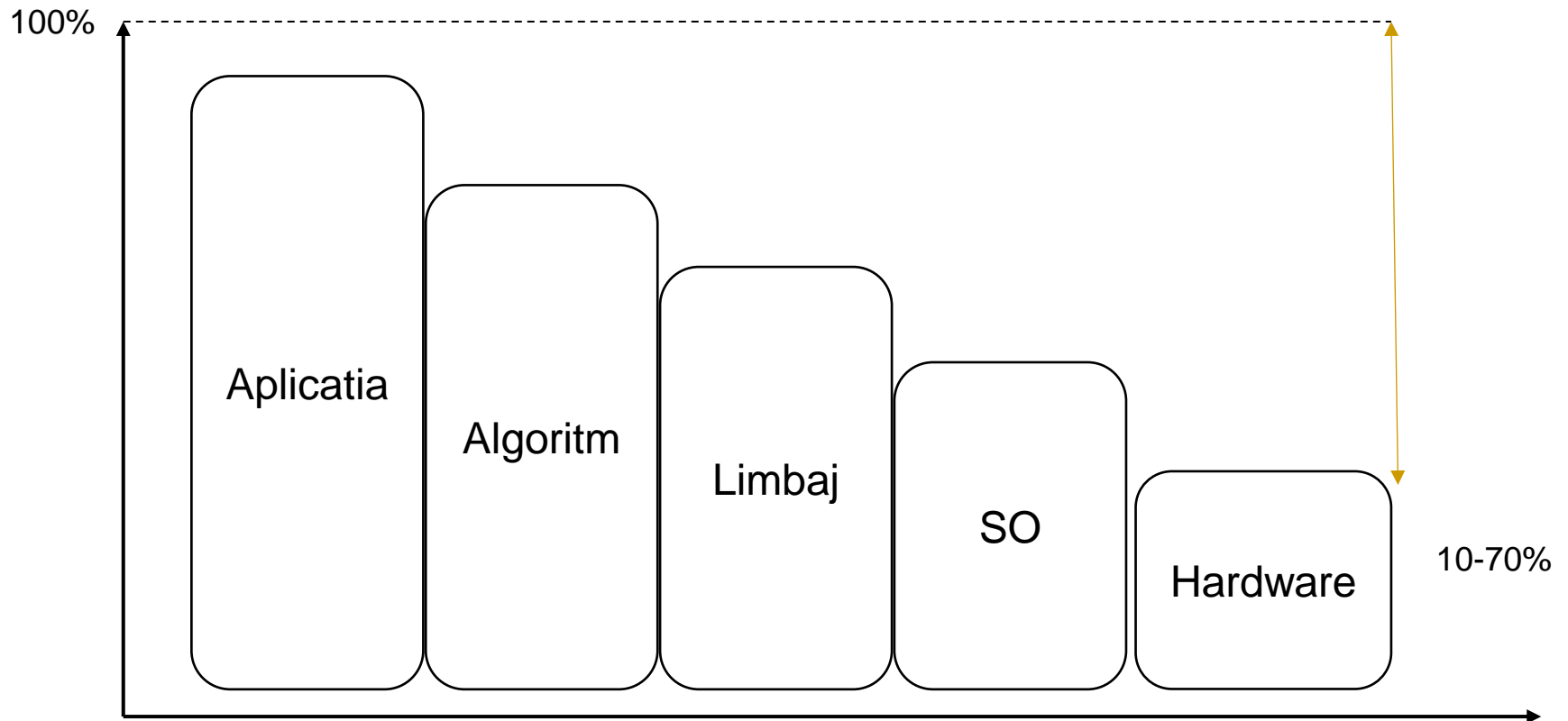
Problematici asociate calculului paralel

Aspecte ale prelucrării 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 si medii de programare 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 $JOB = \{ JOB_1, \dots, JOB_i, \dots, JOB_m \}$

- * task $JOB_i = \{ Ti_1, \dots, Ti_j, \dots, Ti_n \}$

- * proces $Ti_j = \{ Pi_{j1}, \dots, Pi_{jk}, \dots, Pi_{jp} \}$

- * thread $Pi_{jk} = \{ THi_{jk1} \dots THi_{jkt} \}$

- * variabila
- * instructiune
- * bit

Nivelurile de paralelism

Nivel de job-uri

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

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

Nivel de task-uri

- * $JOB_i = \{ Ti_1, \dots, Ti_j, \dots, Ti_n \}$

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

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

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

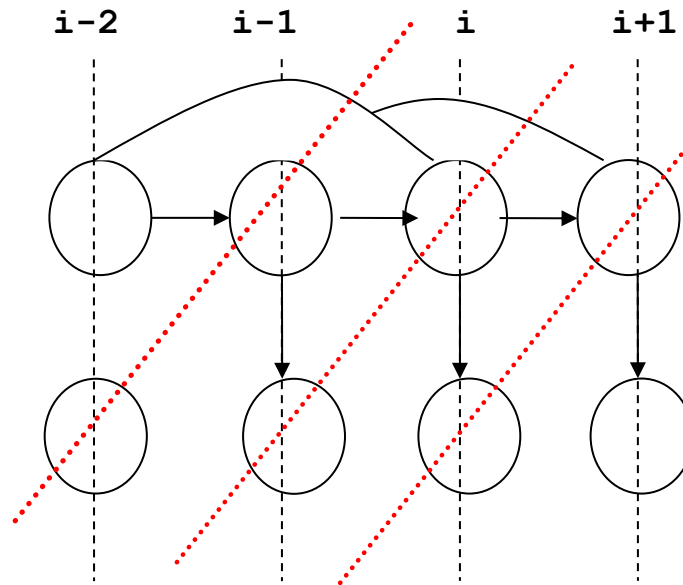
Insa P1(i) si P2(i-1) pot fi efectuate in paralel

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
 $\text{suma}(i) = x(i) + y(i)$
if $\text{sum } a(i) > a$
 then $c = c + 1$
 else $d = d + 1$



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

Nivel de variabila

$P_i = \{ I_{i1}, \dots, I_{ik} \}$

Fiecare proces este format dintr-un 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.

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

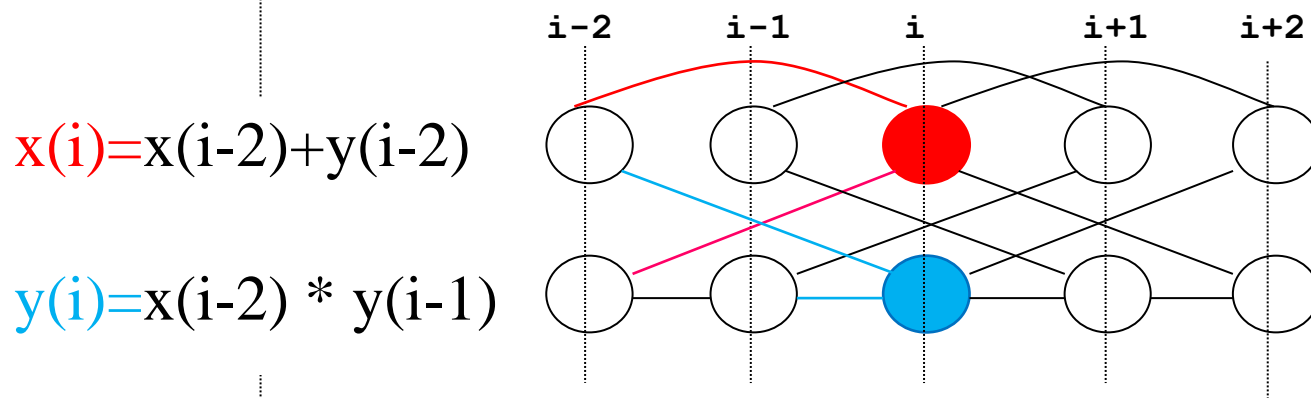
sau

$x(i)$ si $y(i+1)$ se pot calcula in paralel

P1

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

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



Nivel de Instructiune

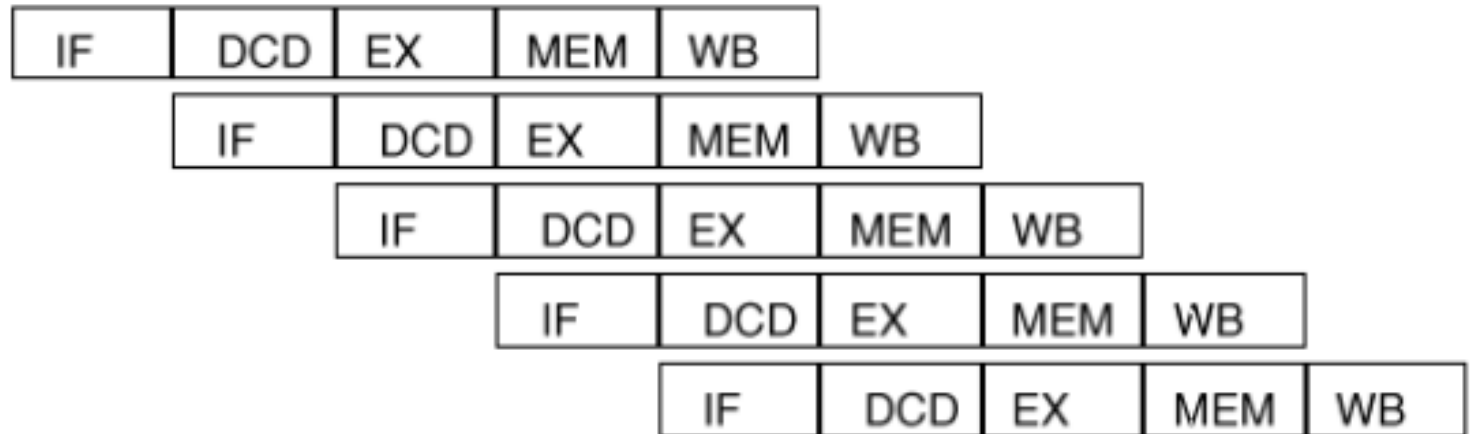
Generarea codului, pentru calculul expresiei $z = x + y$, in structura pipeline

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

- $$200 * (1 + 2 + 3) = 1200$$

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

[illegible]

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

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

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

Sau daca fetch-ul se realizeaza secvential

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

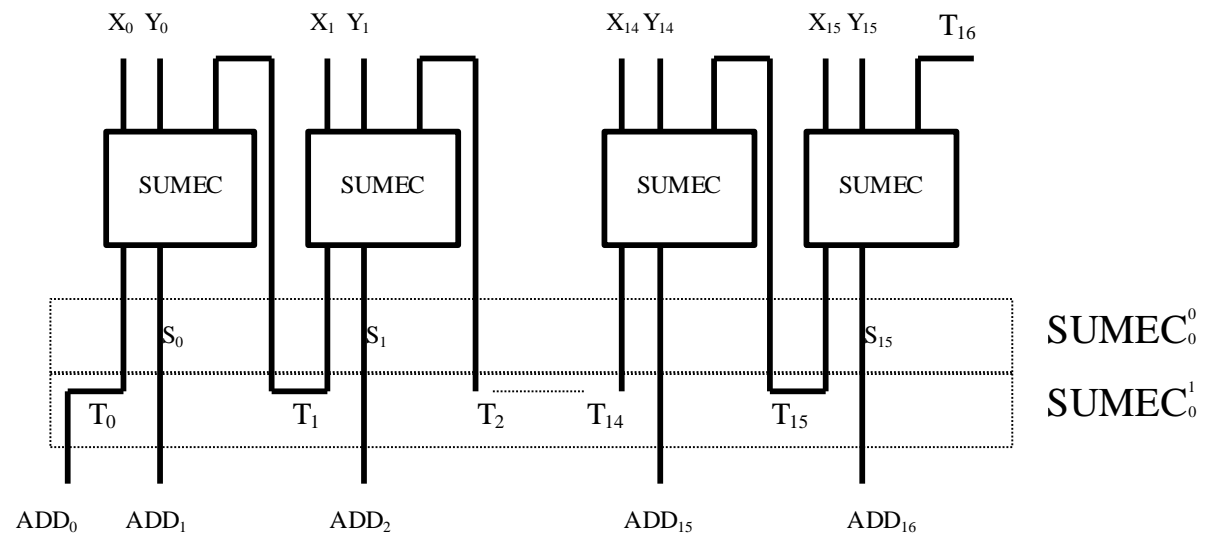
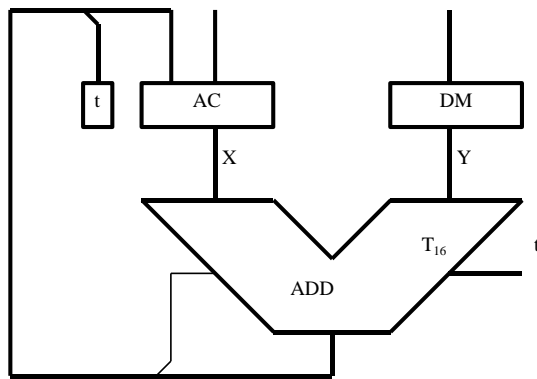
$$1 + 2 + ((n / 2) * 4)$$

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

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 implica 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 compromisuri
- Performanța în sine este o măsură a cât de bine pot fi satisfăcute cerințele de calcul
- 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 sunt capabile „soluțiile” 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 parallel:

- Hardware, rețele, sisteme de operare, biblioteci paralele,
- Limbajele de programare, 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, ar trebui să dureze 10 secunde pe 10 procesoare?
- Mai mulți factori afectează performanța
 - Fiecare factor trebuie înțeles separat
- Dar ei interacționează între ei în moduri complexe
 - Soluția la o problemă poate crea alta
 - O problemă poate masca alta problema, etc.
- Scalarea (dimensiunea problemei) poate schimba condițiile
- Trebuie să înțelegeți limitele de performanță

Calculule paralele

- Un calcul paralel “trivial” este unul care poate fi evident împărțit în task-uri complet independente care pot fi executate simultan
 - Într-un calcul paralel cu **adevărat trivial**, nu există interacțiunea între procese
 - Într-un calcul paralel **aproape trivial** rezultă calcule ce trebuie distribuite și **colectate / combinate** într-un fel

■ Calculele paralele “trivial” 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 adevărat? 






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 **implica unitati functionale hardware** care in general constau din
 - ❑ elemente de procesare;
 - ❑ module de transfer date.
 - ❑ Elemente de stocare
- Ceea ce intereseaza este cum se transpun procesele pe unitatile functionale hardware.

Algoritmi paraleli / Arhitecturi paralele

- H.T.Kung a fost printre 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 modului** care poate fi job, task, proces sau instructiune.
 - De obicei exista posibilitati de paralelism la o granularitate 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 concurrent

- **Control concurrent** 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 (synchronized)** 
- cereri (demand-driven).

exemplu

- algoritmi care prelucreaza matrice se potrivesc foarte bine pe procesoarele sistolice sau pe masive de procesoare iar
- algoritmi 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
 - Datele 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 (T_{seq}) este funcție de
 - dimensiunea problemei și arhitectura
- Runtime paralel (T_{par}) este funcție de
 - dimensiunea problemei și arhitectura paralelă
 - numărul de procesoare utilizate în execuție

■ Performanță paralelă afectată de

- algoritm + arhitectură

■ Scalabilitate

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

Scalabilitate

- Un program 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?
- Se utilizează o măsură de eficiență paralelă
 - Este menținută eficiența pe măsură ce crește dimensiunea problemei?
- Se evaluează valorile 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 productie 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 efectuate 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

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

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.

$$E_p = \frac{V_p}{p} = \frac{T_1}{p * T_p} < 1$$

Eficienta este o masura a eficientei costului

Cost $p * T_p$ este $p * T_p \gg T_1$

Indicatori

Redundanta R_p

Este definita ca raportul intre numarul total al operatiilor

O_p necesar efectuării calculului cu p procesoare si numarul de operatii

O_1 necesar efectuării calculului cu un singur procesor.

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

reflecta timpul pierdut cu overhead-ul.

Utilizarea U_p

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

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



Limite ale calculului paralel

Este foarte important a stabili limita calculului paralel.

Fie T_n timpul necesar executiei a n taskuri de k tipuri diferite.

Fiecare tip consta din n_i taskuri necesitind t_i secunde fiecare

$$T_n = \sum_{i=1}^k (n_i * t_i) \quad \text{iar} \quad n = \sum_{i=1}^k n_i \quad T_n \text{ timpul necesar executarii celor } n \text{ task-uri}$$

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

$$R_n = \frac{n}{T_n} = \frac{n}{\sum_{i=1}^k (n_i * t_i)}$$

$$\text{Fie } f_i = n_i/n \quad \sum_{i=1}^k f_i = 1 \quad R_i = \frac{1}{t_i}$$

In acest caz rezulta

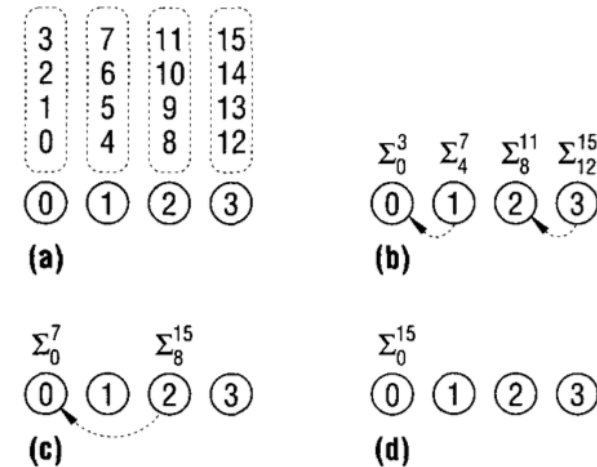
$$R_n = \frac{n}{\sum_{i=1}^k (n_i * t_i)} = \frac{1}{\sum_{i=1}^k (n_i/n * t_i)} = \frac{1}{\sum_{i=1}^k (f_i / R_i)}$$



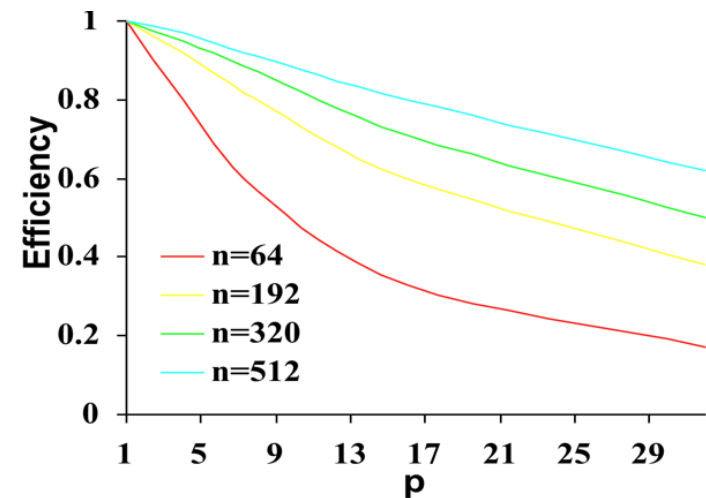
Aceasta relatie reprezinta "bottleneck" - limitarea in structurile paralele.

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:



- $T_p = \frac{n}{p} + 2 * \log p$
- $V_p = \frac{n}{\frac{n}{p} + 2 * \log p}$
- $E_p = \frac{V_p}{p} = \frac{T_1}{p * T_p} = \frac{n}{n + 2 * p * \log p}$
- $E_p = \frac{1}{1 + 2 * p / n * \log p}$



Legea lui Amdahl

Sa consideram

$f = f_{seq}$ procentul de program care se executa secvential

$1-f = f_{par}$ 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

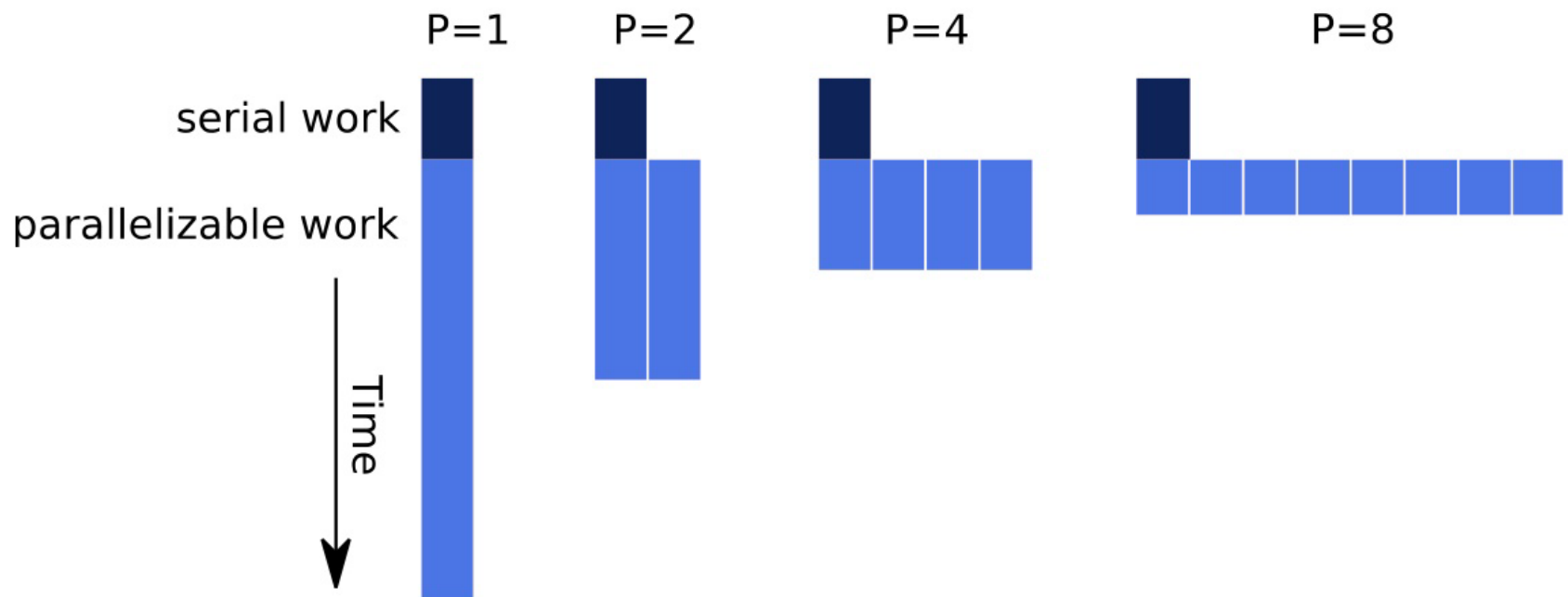
V_p viteza de prelucrare

$$V_p = \frac{T_1}{T_p} = \frac{T_1}{f \times T_1 + \frac{(1-f) \times T_1}{p}} = \frac{1}{f + \frac{(1-f)}{p}} = \frac{1}{f_{seq} + \frac{(1-f_{seq})}{p}} = \frac{1}{f_{seq} + \frac{f_{par}}{p}}$$

Cand $p \rightarrow \infty$

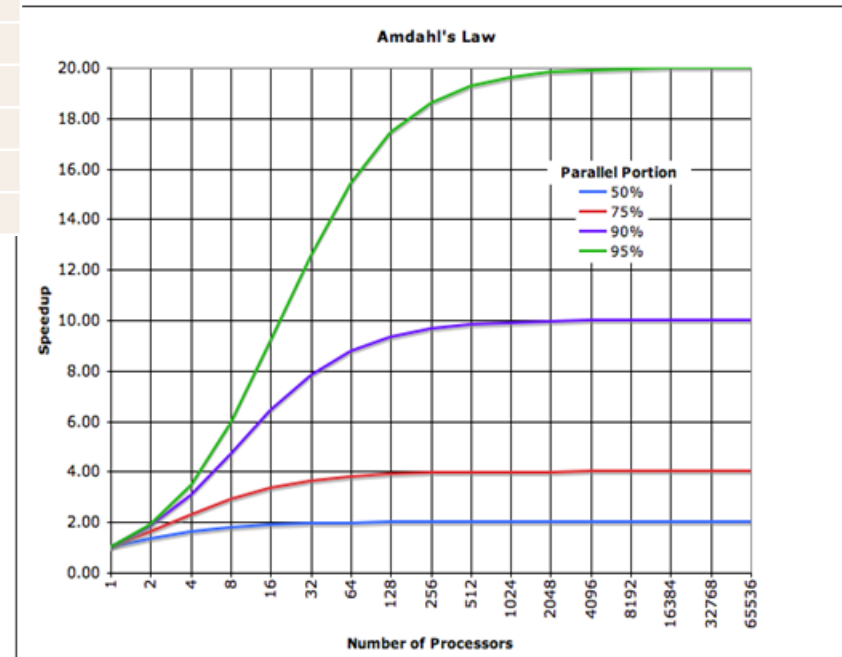
$$V_p = \frac{1}{f} = \frac{1}{f_{seq}}$$

Amdahl



Cresterea in Viteza

	Crestere viteza			
f este % prelucrare secventiala	$f=100\%$	$f=50\%$	$f=10\%$	$f=1\%$
1-f este % prelucrare paralela	$(1-f)=0\%$	$(1-f)=50\%$	$(1-f)=90\%$	$(1-f)=99\%$
Numar procesoare				
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?

$$V_p = \frac{1}{f_{seq} + \frac{f_{par}}{p}}$$

- *Când dimensiunea problemei este fixa*
- *Scalare puternică ($p \rightarrow \infty$, $V_p = \frac{1}{f} = \frac{1}{f_{seq}}$)*

Limita de accelerare este determinată de gradul de secvențialitate, nu de numărul 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

- $T_p = C = (f + (1-f)) * C = (f_{seq} + f_{par}) * C$
- f_{seq} este fracțiunea de T_p pentru execuția secvențială
- f_{par} este fracțiunea de T_p pentru execuție paralelă

Scalarea pe p procesoare conduce la

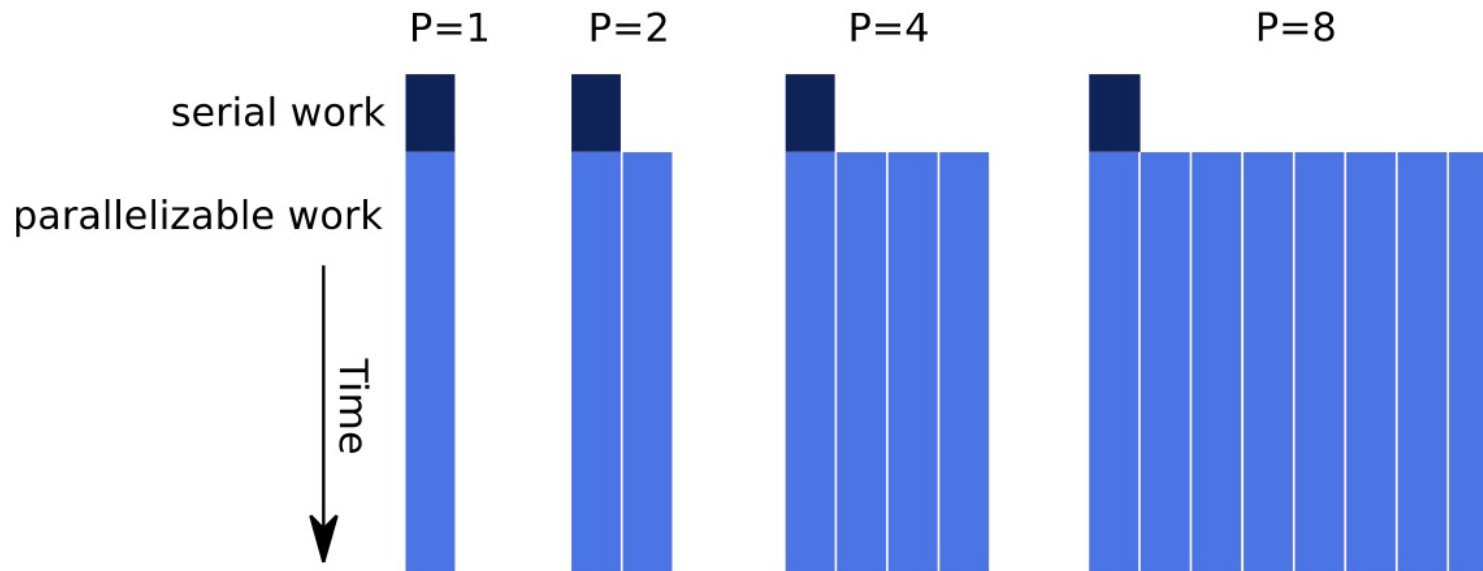
- $T_s = f_{seq} + p * (1 - f_{seq}) = f_{seq} + p * f_{par}$, deoarece $f_{par} = 1 - f_{seq}$ 
- $p * f_{par}$ ne arata cat putem incarca structura din centrul de date
- $T_s = 1 - f_{par} + p * f_{par} = 1 + (p-1)f_{par}$
- $T_p = f_{seq} + (1 - f_{seq}) * p / p = 1$

Care este viteza în acest caz?

$$V_p = T_s / T_p = T_s / 1 = f_{seq} + p(1 - f_{seq}) = 1 + (p-1)f_{par}$$

Gustafson - Baris

Gustafson-Baris



Gustafson-Barsis' / Scalabilitate

Scalabilitate

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

Când se aplică Legea lui Gustafson?

- Când dimensiunea problemei poate crește ca număr
- Scalabilitate slabă $V_p = 1 + (p-1)f_{par}$
- 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 per procesor
- $T_s = fseq + g(p)(1 - fseq)$
 - unde $g(n)$ este creșterea de memorie a unui procesor
- $T_p = fseq + g(p)(1 - fseq) / p$
- $V_p = T_s / T_p = (fseq + g(p)(1 - fseq)) / (fseq + g(p)(1 - fseq) / p)$

Caz 1 : $g(n)=1$

- $V_p = (fseq + 1 - fseq) / (fseq + (1 - fseq) / p) = 1 / (fseq + (1 - fseq) / p) \sim \text{Amdahl}$

Caz 2 : $g(n)=p$

- $V_p = T_s / T_p = (fseq + p(1 - fseq)) / (fseq + p(1 - fseq) / p) = (fseq + p(1 - fseq)) / 1 = fseq + p(1 - fseq) \sim \text{Gustafson}$

Caz 3: $g(p)=m > p$

- $V_p = T_s / T_p = (fseq + m(1 - fseq)) / (fseq + m(1 - fseq) / p) =$
- $(fseq + m(1 - fseq)) / (m/p + (1 - m/p)fseq)$

- Workload-ul crește mai repede decât cererea de memorie.

Control vitezei si limita memoriei

- Metode de control
 - Utilizarea structurilor de date distribuite
 - Utilizarea mecanismelor de caching
 - Utilizarea compresiei de date
 - Utilizarea comunicatiilor asincrone
- Toate metodele de control presupun compromisuri
- Memoria este o resursa critica in cadrul sistemelor de calcul paralel
 - Cerintele de memorie sunt importante pentru designul algoritmilor paraleli eficienti

Limitari de memorie Sun-Ni

■ Baze de date

- ❑ Performanta accesarii BD limitata de memoria sistemului
- ❑ Daca BD nu intra in memorie, citirea de pe disk va scadea timpul total de acces

■ ML

- ❑ Multiple modele ML necesita multa memorie (SVM, RF)
- ❑ Antrenarea in batchuri reduce viteza de antrenare 

■ HPSC

- ❑ CFD, MD necesita multa memorie pentru simulari
- ❑ Daca datele nu intra in memorie, aplicatiile trebuie sa faca checkpointing pe disc crescand timpul de acces si reducand performanta simularii

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

t_s timpul de sincronizare;

t_o 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 :

$$T_{n,p} = t_s + (\lceil N / p \rceil) * (t + t_o)$$

Viteza de prelucrare

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

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 $t_s/(N*t)$ poate fi redus fie prin micsorarea timpului de sincronizare t_s sau prin marirea intervalului intre sincronizari $N*t$
- *efectul overhead-ului* cauzat de t_o/t poate fi redus prin reducerea timpului de overhead t_o 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.

$$\underline{V_{n,p}} = T_1 / \underline{T_{n,p}} = \frac{N*t}{t_s + (\lceil N / p \rceil)*(t+t_o)} = \frac{1}{t_s / (N*t) + (1/N)*(\lceil N / p \rceil)*(1+(t_o/t))}$$

Eficienta

Eficienta

$$En,p = \frac{Vn,p}{p}$$

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

$$p \gggg N \text{ fix}$$

$$Vn,p = \frac{N}{ts/t} \quad si \quad En,p = 0$$

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

$$N \gggg p \text{ fix}$$

$$Vn,p = \frac{N}{ts/t} \quad si \quad En,p = 0$$

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

$$Vn,p = \frac{P}{1 + to/t} \quad si \quad En,p = \frac{1}{1 + to/t}$$

Comparatii Amdahl / Gustafson / Sun & Ni / Worlton

Lege	Scop	Presupuneri	Limitari
Amdahl	Speedup maxim problema fixa (strong scaling)	Divizarea problemei in zone cunoscute (seriale / paralele)	Ignora overheadul de comunicare
Gustafson	Speedup Maxim pentru problema scalabila (weak scaling)	Dimensiunea problemei e scalabila	Ignora overheadul de comunicare
Sun-Ni	Speedup luand in considerare capacitatea memoriei	Divizarea problemei in zona memory-bound & non-memory bound	Presupune overhead de comunicare neglijabil
Worlton	Speedup luand in considerare overheadul de comunicare	Overheadul de comunicare / unitate de lucru si timpul de executie serial sunt cunoscute	Overhead de comunicare proportional cu p si memoria folosita

Comparatie Numerica Amdahl / Gustafson / Sun-Ni / Worlton

■ Lege	<i>Formula</i>	<i>Calcul</i>	<i>Limita</i>
■ Amdahl	$S = 1 / (s + (1 - s) / p)$	$1 / (0.2 + (1 - 0.2) / 16)$	5
■ Gustafson	$S = (1 - s) / s + (1 - s) / p$	$(1 - 0.2) / 0.2 + (1 - 0.2) / 16$	6.88
■ Sun-Ni	$S = p * G(M)$	$16 * G(M)$	$16 * G(M)$
■ Worlton	$S \leq 1 / (1 + (c * p) / T)$	$1 / (1 + (0.1ms * 16) / 100ms)$	5.88
■	Un sistem cu: $p = 16$ procesoare, sectiune seriala $s = 20\%$, timp executie		
■	$T = 100ms$ si overhead de comunicare / unitate de lucru de $c = 0.1ms$		