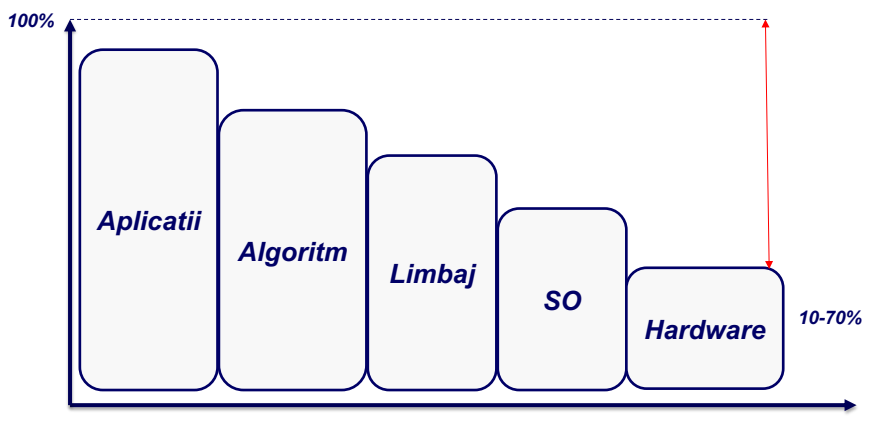
Problematici asociate calculului paralel

Problemele de baza in calculul parallel se refera la urmatoarele aspecte:

- Granularitatea taskurilor
- Elaborarea algoritmilor cu paralelism intrinsec
- Proiectarea limbajelor de programare
- Proiectarea unor arhitecturi hardware adecvate
- Proiectarea unor sisteme de operare adecvate

Gradul de Paralelism



Nivelurile de Paralelism

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

- Job — $JOB = \{ JOB_1, \dots, JOB_i, \dots\dots, JOB_m \}$
- Task — $JOB_i = \{ Ti_1, \dots, Ti_j, \dots\dots, Ti_n \}$
- 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 – Problema

- **Problema**
  - Un robot cu mai multe grade de libertate
  - Pentru fiecare grad de libertate exista un procesor
  - Programul de conducere al robotului este partitionat in task-uri care se ocupa de cite un grad de libertate al robotului
  - Taskurile se executa in paralel insa **interactioneaza** pentru miscarea robotului
- **Nivelul de Job** – Gestionarea robotului
  - $JOB = \{ JOB_1, \dots, JOB_i, \dots\dots, JOB_m \}$
- **Nivelul de Task** – Gestionarea joburilor robotului
  - $JOB_i = \{ Ti_1, \dots, Ti_j, \dots\dots, Ti_n \}$
  - Fiecare Ti se executa pe cate un procesor distinct
  - Exista o relatie intre Ti si Tj pentru transferul de date / completarea functiilor de prelucrare realizate in cadrul job-ului.

Nivelul de Task

- $T_j = \{Pi_{j1}, ...,Pi_{jk}, ..., Pi_{jp}\}$
- Task-urile sunt alcatuite din procese identificate de utilizator sau compilator pentru structuri multiprocesor

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
```

```
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
```

are doua procese P1 si P2

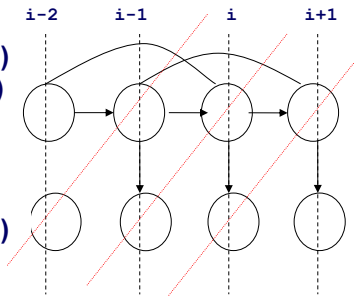
- Intre P1(i) si P2(i) exista o **dependenta de date**, deci nu pot fi efectuate in paralel
- Insa **P1(i)** si **P2(i-1)** pot fi efectuate in paralel

Nivelul de Proces

- Interdependenta intre procese este

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)

Nivelul de Thread

- Pi_{jk} ={THi_{jk1} … THi_{jkt} }
- Procesele se mapeaza peste threadurile de la nivel de SO
- Runtime-ul SO decide modul de executie al threadurilor
- Sistemul de scheduling face maparea threadurilor utilizator peste cele SO

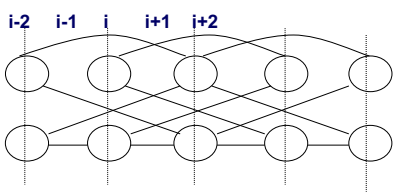
Nivelul de Variabila

- Pi= {I_{i1}, ….I_{ik}}
  - Fiecare proces este format dintr-o multime de instructiuni care calculeaza variabilele de iesire in functie de variabilele de intrare
  - Paralelismul in cadrul unui proces se realizeaza prin calculul simultan a mai multe variabile de iesire
  - Putem considera ca in cadrul procesului P_1 variabilele
    - x(i) si y(i) se pot calcula in paralel
    - x(i) si y(i+1) se pot calcula in paralel

P_1
x(i) = x(-2) + y(i-2)
y(i) = x(i-2) * y(i-1)

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

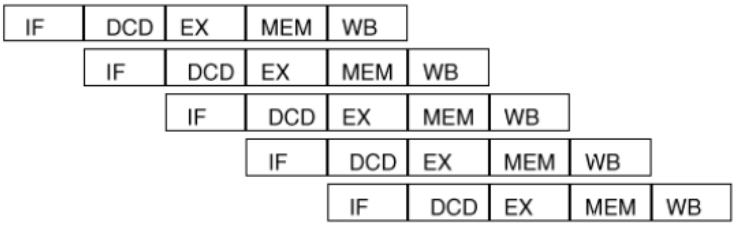
y(i) = x(i-2) * y(i-1)



Nivelul 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



Problema

- Consideram un procesor ce 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 ca intr-un ciclu de instrucțiune:
  - Citirea necesita **o singura** perioada de ceas
  - Decodarea necesita **două perioade** de ceas
  - Executia necesita **trei** perioade de ceas
- In total avem o secvență de **200 de instrucțiuni**

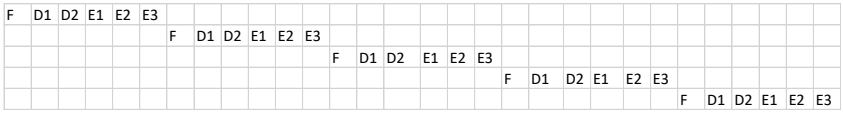
84

Solutie Simpla + Pipeline Scalara

- O structure fara pipeline necesita 1200 de cicli de ceas: (n * 6)

  200 * (1 + 2 + 3) = 1200

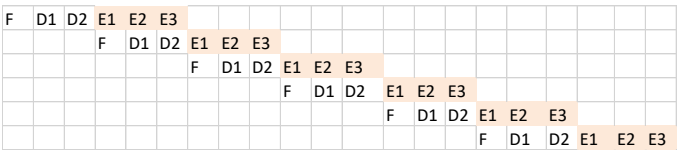
  n * (1 + 2 + 3)



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

  1 + 2 + (200 * 3) = 603

  1 + 2 + (n * 3)



84

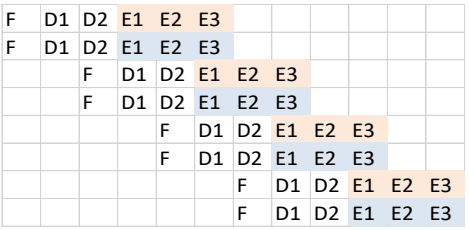
85

Solutie pipeline Superscalara x2

- Pipeline superscalar cu două unități paralele necesita 303 cicli de ceas (n / 2 + 1) * 3)

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

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

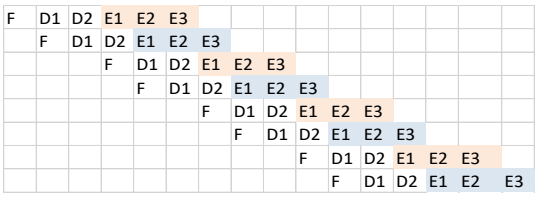


- Daca fetch-ul se realizeaza secvential

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

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

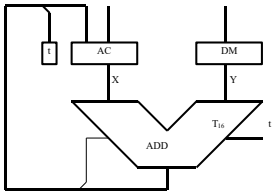
  1 + 2 + ((n / 2) * 4)

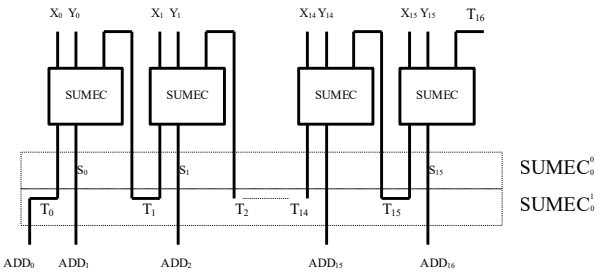


85

Nivelul de Bit

- Toate procesoarele utilizeaza unitati aritmetice paralele
  - Cu/fara anticiparea transportului
  - Cu/fara structura pipeline





Ce este Performanța Paralelă?

- Performanța paralela a unui sistem de calcul implica doi factori
  - Cerințele de calcul (ce trebuie făcut)
  - Resursele de calcul (cat costă să o faci)
- Problemele de calcul se traduc în cerințe de **cod/algoritmi**
- Resursele de calcul: **interacțiune** și **compromisuri** intre componentele sistemului
- **Performanța** este măsura a cât de bine pot fi satisfăcute cerințele de calcul
- **Evaluăm performanța** pentru a **înțelege relațiile** dintre cerințe și resurse
- Decidem cum "**schimbam soluțiile**" pentru a atinge obiectivele
- **Măsurile de performanță** reflectă deciziile despre **cum** și **cât de bine** sunt capabile „soluțiile" să satisfacă cerințe de calcul

88

Probleme de Performanță

- Când folosim un sistem paralel de calcul
  - Performanța este rațiunea de “a fi” pentru paralelism
  - Performanță vs. eficienta sistemului paralel de calcul
- Performanță paralelă vs. secvențială
- Fara **performanță** paralelismul este inutil…
- Prelucrarea în paralel include tehnici și tehnologii necesare pentru calculul în parallel:
  - Hardware, rețele, sisteme de operare, biblioteci paralele
  - Limbajele de programare, compilatoare, algoritmi, profiling, ...
- Paralelismul trebuie să ofere performanță
- Cum? Cât de bine?
  - Trebuie sa introducem / definim metrici si unitati de masura

88

89

Limitele Calcului Paralel

- Dacă fiecare procesor este evaluat la
  - k GFLOPS și există p procesoare,
- Ar trebui să vedem performanța k * p GFLOPS?
  - Dacă durează 100 de secunde la un procesor, ar trebui sa dureze 10 secunde pe 10 procesoare?
- Mai multi 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 (dimensionarea problemei) poate schimba performantele rezolvarii problemei
- Trebuie să înțelegem limitele calculului parallel
  - Performanta vs. eficienta

89

90

Embarassingly Parallel

- Un calcul “trivial” paralel este unul care poate fi împărțit în task-uri complet independete ce pot fi executate simultan
  - Într-un calcul **trivial paralel** nu există interacțiune între procese
  - Într-un calcul **aproape trivial paralel** există calcule ce trebuie distribuite și colectate / combinate într-un fel
- Calculele **trivial paralele** au potențial pentru a atinge performanta maximă pe sisteme de calcul paralele
  - Timpul T secvențial poate fi redus la T / P ruland în paralel cu P procesoare
  - De ce acest lucru nu este întotdeauna adevarat?

90

91

Algoritmi / Arhitecturi Paralele

- Prelucrarea paralela include algoritmi si arhitecturi paralele
- Un algoritm paralel poate fi considerat ca o colectie de procese independente ce se executa simultan, procesele comunicand in timpul executiei
- Un algoritm se executa pe unitati functionale hardware care in general constau din
  - Elemente de procesare
  - Module care transfer date
- Ceea ce intereseaza este cum se transpun eficient procesele pe unitatile functionale hardware

91

92

Algoritmi / Arhitecturi Paralele v2

- H.T.Kung propune corelarea Algoritmi/Arhitectura:

Algoritmi paraleli	Arhitecturi paralele
Granularitate	Complexitate procesor
Control concurrent/sincronizare	Mod de operare
Transferul datelor	Structura memoriei
Geometria comunicatiei	Retele de comutare
Complexitate algoritm	Numar de procesoare
Dependenta taskuri	Dimensiune memorie

92

93

Granularitate / Complexitate

- Granularitatea obiectelor se refera la complexitatea modulului care poate fi job, task, process, thread sau instructiune
  - De obicei exista posibilitati de paralelism la o granularitate mare dar care nu este exploatata deoarece implica o comunicatie intensa si o crestere a complexitatii software-ului dezvoltat
  - La acest nivel se face o analiza intre granularitate si comunicatie pentru a stabili solutia cea mai buna / eficienta

93

94

Control Conturent / Operare

- Control concurent se refera la schema de selectie a modulelor pentru executie
  - Trebuie sa satisfaca dependenta de date si control
  - Asigurarea excluderii mutuale a accesului la resurse
- Schemele de control sunt bazate pe:
  - Disponibilitatea datelor (data flow)
  - Controlul centralizat (sinchronized)
  - Cererile de acces (demand-driven)
- Exemple:
  - Algoritmii ce proceseaza matrice se potrivesc pe procesoare sistolice
  - Algoritmii ce utilizeaza transferuri conditionate / iregularitati se potrivesc pe arhitecturi asincrone / data-flow

94

95

Transfer / Geometrie / Complexitate

- Transferul datelor se refera utilizarea operanzilor
  - 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
- Complexitatea algoritmilor se refera la numarul de operatii necesare pentru implementarea unui algoritm
  - Influenteaza direct numarul de procesoare si dimensionarea memoriei necesare / utilizate

95

Performanță și Scalabilitate

- Evaluare
  - Runtime secvențial (T_{seq}) este o funcție de
    - Dimensiunea problemei și arhitectura secventiala utilizata
  - Runtime paralel (T_{par}) este o funcție de
    - Dimensiunea problemei și arhitectura paralelă utilizata
    - Numarul de procesoare utilizate în execuție + modul de interconectare
- Performanță paralelă este afectată atat de algoritm cat si de arhitectura
- Scalabilitatea este:
  - Abilitatea algoritmului paralel de a **creste performanța proporțional** cu **numărul de procesoare** și cu **dimensiunea problemei**

Strong vs. Weak Scaling

- Strong scaling
  - Masoara cum timpul de executie **scade** odata cu **cresterea numarului de procesoare** pentru o **dimensiune fixa** a problemei
  - Este o masura a paralelismului intrinsec dintr-o problema
  - Foarte util pentru probleme cu granularitate crescuta
    - Taskuri independente ce pot fi executate simultan

98

Exemplu de Strong Scaling

- Presupunem ca un cod parallel rezolva o problema in 100s folosind un processor
- Daca folosim 2 procesoare si programul necesita 50s pentru rezolvare acesta a atins scalabilitate “**puternica**” perfecta
  - Programul a putut fi spart in subprobleme complet independente
  - Fiecare processor si-a putut executa taskul in jumatate din timpul necesar initial

98

99

Strong vs. Weak Scaling

- Weak scaling
  - Masoara cum timpul de executie **creste** odata cu **cresterea dimensiunii problemei** si a **numarului de procesoare**
  - **Incarcarea pe fiecare processor ramane constanta**
  - Este o masura a scalabilitatii unui cod la dimensiuni crescute

99

100

Exemplu de Weak Scaling

- Presupunem ca un cod parallel poate rezolva o problema de dimensiune 1M in 100s folosind un processor
- Daca vom creste dimensiunea problemei la 2M si vom utiliza 2 procesoare iar timpul de procesare ramane 100s programul atinge scalabilitate "**slaba**" perfecta
  - Programul a scalat la dimensiuni mai mari fara nicio pierdere de performanta

100

101

State of Play

- In practica, e rar ca un program paralel sa atinga scalare perfecta "strong" sau "weak"
- Masurand scalabilitatea strong/weak putem identifica zone in care putem imbunatati programele noastre
- Ce tip de scalabilitate conteaza mai mult, depinde de natura problemei/aplicatiei
  - Daca e importanta **scaderea timpului total de rezolvare** – **strong scaling** este mai relevant
  - Daca e importanta **rezolvarea unei probleme mari** ce **nu poate fi rezolvata de un singur processor** – **weak scaling** este mai relevant

101

102

Strong/Weak Scaling Powerups

- Evitati zonele secventiale de cod
- Folositi algoritmi eficienti de comunicare
- Utilizati algoritmi eficienti de load-balancing
- Echilibrati incarcarea wordloadurilor intre procesoare
- Realizati un pas de tunare / optimizare al executiei aplicatiilor paralele pentru hardware-ul specific pe care se realizeaza rularea

102

103

Scalabilitate vs. Eficienta

- 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 utilizeaza o măsură de **eficiență paralelă**
  - Este menținută eficiența pe măsură ce crește dimensiunea problemei?
- Cum se evalueaza valorile de performanță?

103

Indicatori de Performantele ai Calculului Paralel – v1

104

- **Rata de executie** masoara rata de producere a unor rezultate in unitatea de timp a unei utitati single/multiprocesor
  - MIPS / GIPS / TIPS – milioane/giga/tera de instructiuni pe secunda
    - Inadecvat SIMD
  - MOPS / GOPS / TOPS – milioane/giga/tera de operatii pe secunda care masoara operatiile efectuate de unitatile de prelucrare
    - Ignora lungimea cuvantului si natura operatiilor
  - MFLOPS / GFLOPS / TFLOPS / PFLOPS – milioane /giga/tera/peta de operatii cu virgula mobila pe secunda
    - Util in zona numerica / putin relevant in zona alphanumerica sau ML/AI
  - MLIPS / GLIPS/ TLIPS – milioane /giga/tera de inferente logice pe secunda
    - Utile in aplicatii ML/AI

104

Indicatori de Performantele ai Calculului Paralel – v2

105

- Avem o structura paralela cu **p** procesoare
- **Viteza de prelucrare:** $V_p = \frac{T_1}{T_p}$ unde
  - T_1 – timpul pentru prelucrarea pe un procesor
  - T_p – timpul pentru prelucrarea pe p procesoare
  - V_p = raportul intre prelucrarea secventiala si cea paralela
  - Deoarece exista overhead cu sincronizarea: $1 \leq V_p < p$
- **Eficienta:** $E_p = \frac{V_p}{p} = \frac{T_1}{p * T_p} < 1$ unde
  - Eficienta este ca raportul dintre viteza de prelucrare si numarul de procesoare
  - Este o masura a eficientei costului
  - Cost p* T_p este p* T_p >> T_1

105

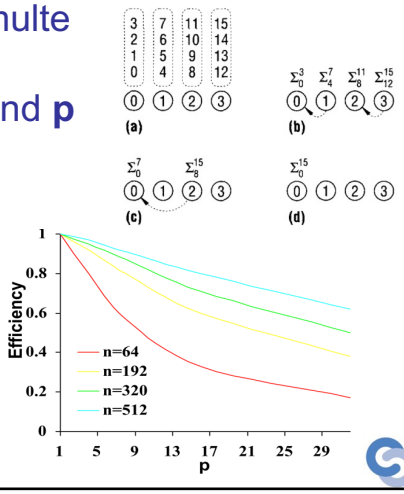
Indicatori de Performantele ai Calculului Paralel – v3

- **Redundanta:** $R_p = \frac{O_p}{O_1} =$ $\frac{\text{nr total de operatii efectuate pe cele p procesoare}}{\text{nr de operatii necesare efectuarii pe 1 procesor}}$ unde
  - **R_p** este raportul intre numarul total al operatiilor **O_p** necesar efectuarii calculului cu p procesoare si numarul de operatii **O_1** necesar efectuarii calculului cu un singur procesor
- **Utilizarea:** $U_p = \frac{O_p}{p * T_p} \leq 1$ unde
  - **U_p** este 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**

106

Exemplu – Scalabilitatea adunarii a 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:
- $T_p = \frac{n}{p} + 2 * \log p$
- $V_p = \frac{T_1}{T_p} = \frac{n}{\frac{n}{p} + 2 * \log p}$
- $E_p = \frac{V_p}{p} = \frac{T_1}{p * Tp} = \frac{n}{n + 2 * p * \log p}$
- $E_p = \frac{1}{1 + 2 * p/n * \log p}$

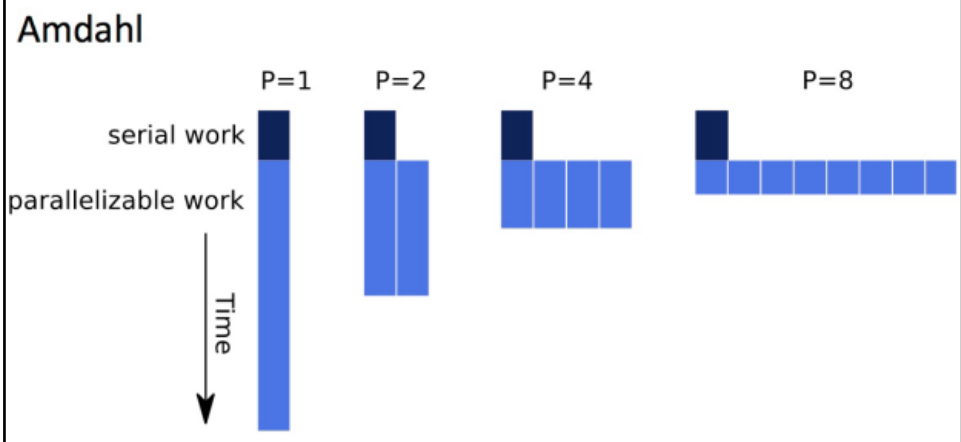


107

Legea lui Amdahl

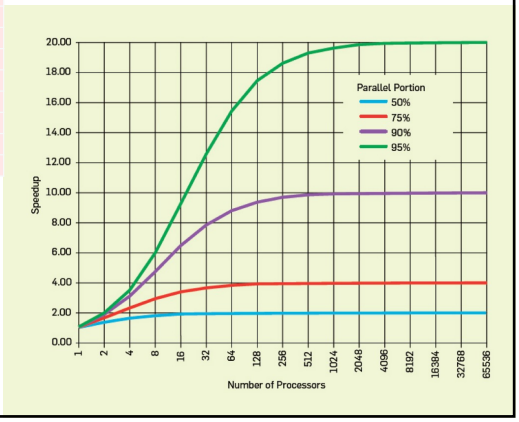
- Sa consideram
  - f = f_{seq} procentul de program care se executa secvential
  - 1-f = f_{par} procentul de program care poate fi paralelizat
- Fie T_1 timpul de executie pe o structura cu 1 procesor
- Fie T_p timpul de executie pe o structura cu p procesoare
- Viteza de prelucrare
  - $V_p = \frac{T_1}{T_p} = \frac{T_1}{f*T_1+(1-f)*T_1/p} = \frac{1}{f+(1-f)/p)}$
  - $V_p = \frac{1}{f_{seq}+f_{par}/p} = \frac{1}{f_{seq}+(1-f_{seq})/p}$ cu $p \rightarrow \infty$
  - Astfel $V_p = \frac{1}{f} = \frac{1}{f_{seq}}$

Legea lui Amdahl – v2



Scalabilitatea Legii lui Amdahl – v1

	Crestere viteza			
f este % prelucrare **secventiala**	*f=100%*	*f=50%*	*f=10%*	*f=1%*
1-f este % prelucare **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



Scalabilitatea Legii lui Amdahl – v2

- Capacitatea algoritmului paralel de a obține speedup e **proporțional cu numărul de procesoare** și **dimensiunea problemei**
- Când se aplică Legea lui Amdahl? $V_p = \frac{1}{f_{seq} + f_{par}/p}$
  - Când **dimensiunea problemei este fixa**
  - **Scalare puternică** ($p \to \infty$, $V_p \to 1 / fseq$)
- Speeup-ul este limitat de **partea seriala nu de numarul de procesoare**!
- Eficiența perfectă (strong) este greu de realizat!

Legea lui Gustafson-Barsis – v1

- Speedup pentru o problema scalata – perspectiva unui DataCenter
  - Probleme mari & utilizarea eficienta a resurselor
  - Problema poate fi scalata (modificarea dimensiunii!)
- Cât de mare poate fi o problema? (HPL/MD/QED)
- Care este contrangerea problemei care se executa in parallel?

Presupunem că timpul de executie este constant

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

Legea lui Gustafson-Barsis – v2

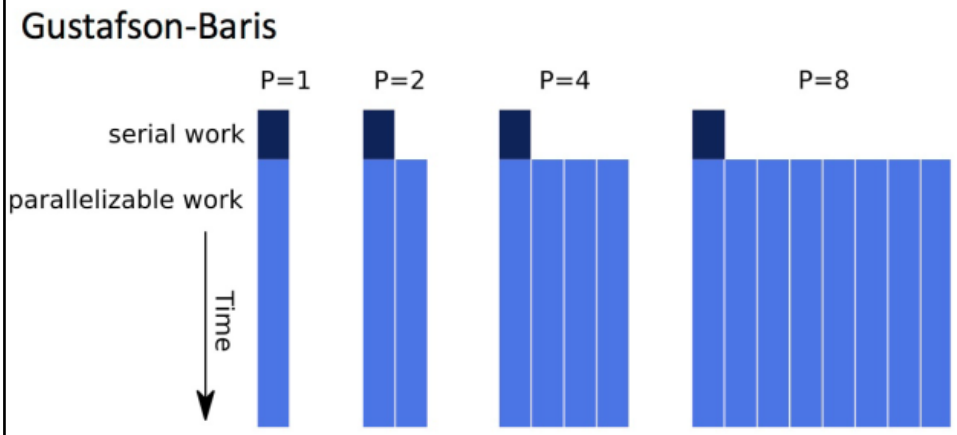
Scalarea pe **p** procesoare conduce la

- $T_s = f_{seq} + p*(1- f_{seq}) = f_{seq} + \mathbf{p*f_{par}}$, deoarece $f_{par} = 1- f_{seq}$
- $\mathbf{p*f_{par}}$ ne arata incarcarea centrului de date
- $T_s = 1- f_{par} + p*f_{par} = \mathbf{1 + (p-1)f_{par}}$
- $T_p = f_{seq} + (1-f_{seq})*p / p = 1$

Care este speedup-ul în acest caz?

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

Legea lui Gustafson-Barsis – v3



Scalabilitate Gustafson-Barsis

Scalabilitate

- Capacitatea algoritmului paralel de a scala este **proporțional cu numărul de procesoare** și cu **dimensiunea problemei**
- Când se aplică Legea lui Gustafson?
  - Când dimensiunea problemei poate fi crescuta
  - Scalabilitate slabă (weak scaling) $V_p = 1 + (p-1)f_{par}$
  - Speedup-ul include si numărul de procesoare!
  - Cand se poate menține sau crește eficiența paralelă prin scalarea problemei

Comparatie Amdahl / Gustafson

- Considerand s fractiunea de cod serial f_{seq} de mai devreme si p procesoare
  - Legea lui Amdahl este Speedup = $V_p = \frac{1}{s + (1-s)/p}$
  - Legea lui Gustafson este Speedup = $V_p = \frac{1-s}{s + (1-s)/p}$
- Exemplu: daca s = 20% si regiunea de cod paralelizabil este astfel 80% cele doua legi indica:

Numar procesoare	*Speedup Legea lui Amdahl*	*Speedup Legea lui Gustafson*
2	1.6	2.5
4	2.2	4
8	2.6	5
16	2.9	6

- Gustafson prezice speedup-uri mai mari

Discutie Amdahl / Gustafson

- Care lege e mai “precisa”?
  - In practica, estimarea legii lui Gustafson este mai precisa decat legea lui Amdahl pentru ca majoritatea problemelor sunt scalabile
    - Se poate creste dimensiunea problemei pentru a folosi puterea de calcul
  - Atat Amdahl cat si Gustafson sunt limite teoretice!
- Speedup-ul real e mereu mai mic si depinde de
  - Eficienta algortimului parallel
  - Overheadul parallel
  - Dimensiunea problemei (weak scaling)
  - Dimensiunea sistemului de calcul (strong scaling)
- Legile ne ajuta sa intelegem limitarile si beneficiile sistemelor paralele de calcul