



# Arhitectura Sistemelor de Calcul



**Universitatea Politehnica Bucuresti**  
**Facultatea de Automatica si Calculatoare**

[cs.ncit.pub.ro](http://cs.ncit.pub.ro)

[curs.cs.pub.ro](http://curs.cs.pub.ro)



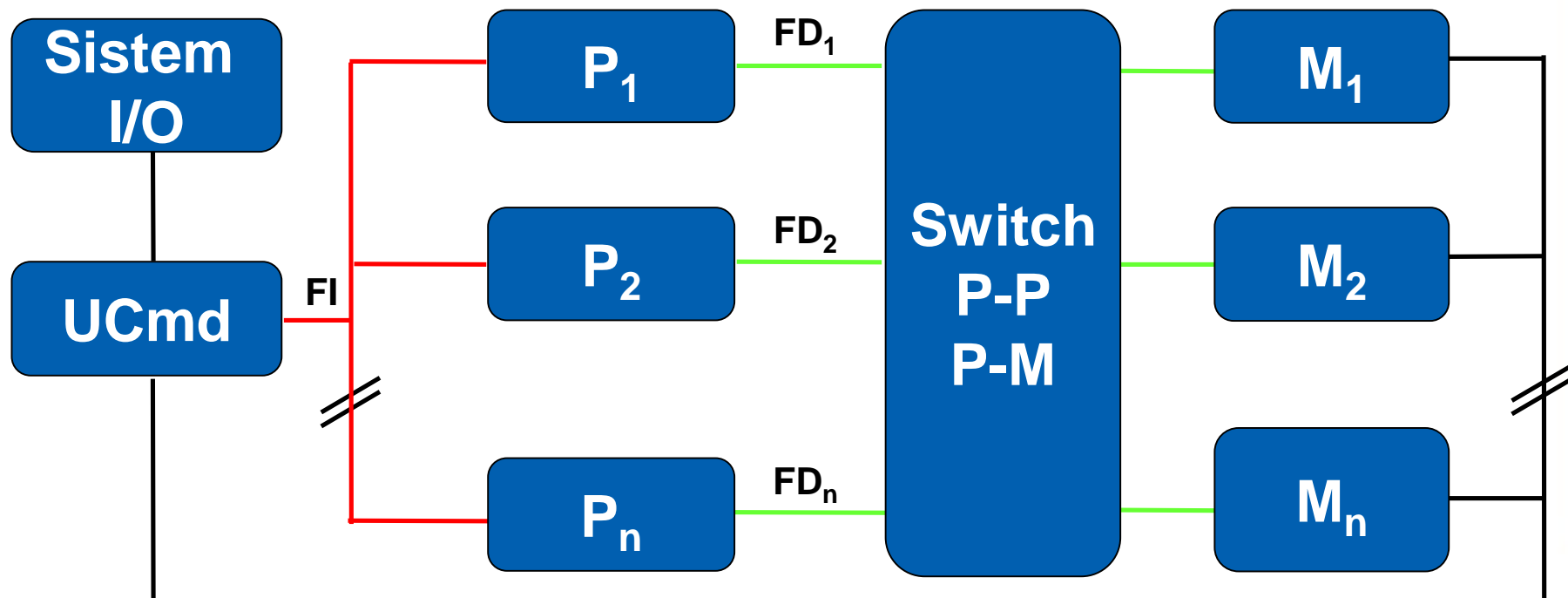
- Structura SIMD
- Setul de Instructiuni Masina SIMD
- Implementarea Salturilor Conditionate in Structura SIMD
- Organizarea Datelor in Structura SIMD



# Structura SIMD

3

- Arhitecturile SIMD sunt adecvate prelucrarilor vectoriale:
  - O singura UCmd
  - Mai multe procesoare aritmetice si module de memorie
  - Retea de comutatie P-P & P-M
- Structuri adecvate prelucrării vectoriale → esentiala organizarea algoritmului de calcul
- Pentru aplicatii neorientate pe aplicatii vectoriale → ineficiente





# Structura SIMD - Componente

4

- **UCmd:** Citeste, interpreteaza si executa instructiuni de control
  - Trebuie sa aiba registre generale rapide, unitate de prelucrare si memorie locala
  - Executa si instructiunile de ramificatie (nu ca  $P_i$ )
- **$P_i$ :** Sunt sincronizate la nivel de instructiune → fiecare procesor lucreaza pe un alt set de date
  - Trebuie sa aiba registre locale
- **M:** Formata din n module
  - Contine instructiuni (de comanda & prelucrare) si date



# Structura SIMD - Componente

5

- **Instructiuni:**
  - De control → executate pe UCmd
  - De prelucrare vectoriala → executate de  $P_i$
  - Fluxul de instructiuni e similar cu cel din sisteme secventiale
- **Registre index:**
  - Globale → in UCmd; afecteaza cu aceeasi cantitate adresa operanzilor
    - Accesul pe linii
  - Locale → in fiecare  $P_i$ ; pot fi independente si se refera la zona de memorie  $M_i$  asociata fiecarui procesor
    - Acces pe coloane
    - Altfel se poate ajunge la secventialitatea accesului la date!



- Structura SIMD
- Setul de Instructiuni Masina SIMD
- Implementarea Salturilor Conditionate in Structura SIMD
- Organizarea Datelor in Structura SIMD



# Exemplu – Inmultirea de Matrice

7

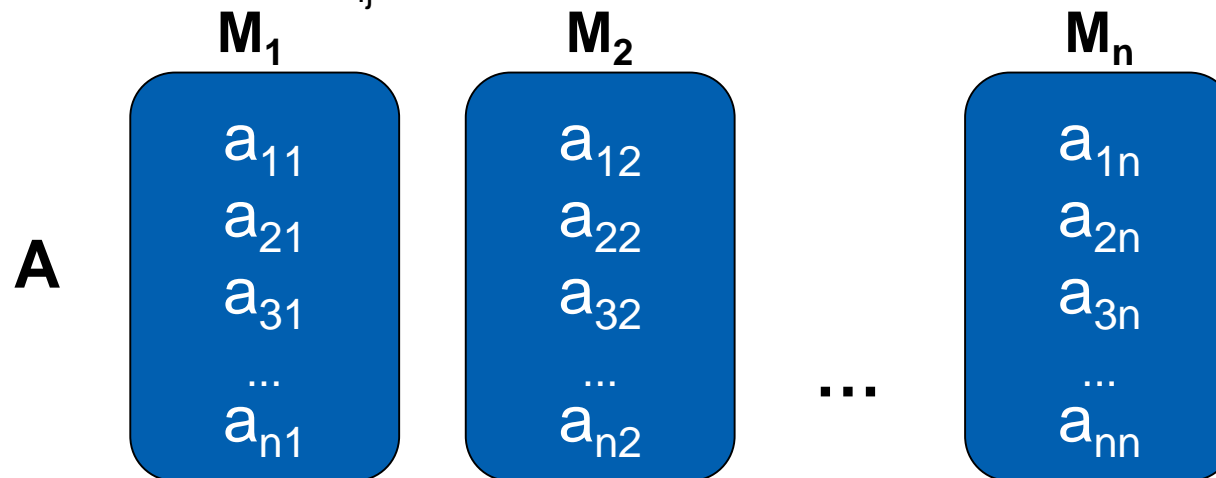
- Problema:  $A[n, n]$ ,  $B[n, n]$ ,  $C = A \times B$
- Considerand  $(0 \leq k \leq n - 1) \rightarrow$  se opereaza pentru toti indicii  $k$  simultan, adica se calculeaza pe linii

**for**  $j = 0$  **to**  $n - 1$

$$c_{ik} = c_{ik} + a_{ij} * b_{jk} \quad (0 \leq k \leq n - 1)$$

**end j loop**

- Elementele unei linii sunt in module diferite de memorie!
- Aceeasi discutie cu  $a_{ij}$  care nu depinde de  $k$ ! (Curs 5)



- Matricele  $B$  &  $C$  sunt aranjate similar
- Linia rezultat:  $c_{i1}, c_{i2} \dots c_{in}$  este calculata in paralel pe  $P_1, P_2 \dots P_n$



# Setul de Instructiuni Masina SIMD

8

- $P_i$  are un registru acumulator  $ACC_i \rightarrow$  info locala; **UCmd** are un registru index  $INDEX[i]$
- **Instructiuni masina cu vectori:** ( $0 \leq k \leq n - 1$ )
  - Incarcare:
    - vector:  $LOAD\ A \quad ;ACC[k] \leftarrow A[0,k]$  (pentru vectori e suficient  $A[k]$ )
    - vector indexat:  $LOAD\ A[i] \quad ;ACC[k] \leftarrow A[INDEX[i],k]$
  - Memorare:
    - vector:  $STO\ A \quad ;A[0,k] \leftarrow ACC[k]$
    - vector indexat  $STO\ A[i] \quad ;A[INDEX[i],k] \leftarrow ACC[k]$
  - Adunare vector:  $ADD\ A[i] \quad ;ACC[k] \leftarrow ACC[k] + A[INDEX[i],k]$
  - Inmultire vector:  $MUL\ A[i] \quad ;ACC[k] \leftarrow ACC[k] * A[INDEX[i],k]$
  - Difuzie scalara:  $BCAST\ i \quad ;ACC[k] \leftarrow ACC[INDEX[i]]$  (1 perioada de ceas)
- $[INDEX[i], k]$  e organizarea pe linii;  $[k, INDEX[i]]$  e organizarea pe coloane
- Instructiunile sunt similare SISD (von Neumman) cu diferenta k-ului ce indica paralelismul





# Instructioni de Ciclare & Comparare

9

- Instructioni referitoare le registre INDEX:
  - Incarcare:
    - ENXC i, ct                      ;INDEX[i]  $\leftarrow$  ct
    - LDNX i, Adr                      ;INDEX[i]  $\leftarrow$  MEM[Adr]
  - Incrementare:
    - INCX i, ct                      ;INDEX[i]  $\leftarrow$  INDEX[i] + ct
  - Compara index & salt la adresa:
    - CPNX i, j, Adr                      ;if INDEX[i] < INDEX [j]  
   ;                      then Jump Adr  
   ;else continue
- Adresele de genul A[i], B[j] sunt adrese indexate cu continutul unui registru INDEX de pe UCmd



# Exemplu - Inmultire de Matrice

10

```
for j = 0 to n - 1
```

```
     $c_{ik} = c_{ik} + a_{ij} * b_{jk} \quad (0 \leq k \leq n - 1)$ 
```

```
end j loop
```

- j si n sunt incarcate in doua registre INDEX
- i provine din bucla mare a inmultirii

	ENXC j, 0	;INDEX[j] = 0
	LDNX n, adrN	;la adrN se afla valoarea n
adr:	LOAD A[i]	;ACC[k] ← A[INDEX[i],k]
	BCAST j	;ACC[k] ← ACC[INDEX[j]] (f important!)
	MUL B[j]	;ACC[k] ← ACC[k] * B[INDEX[j],k]
	ADD C[i]	;ACC[k] ← ACC[k] + C[INDEX[i],k]
	STO C[i]	;C[i] ← ACC[k] (depunere rezultat)
	INCX j, 1	;INDEX[j]++
	CPNX j, n, adr	;salt la "adr" cand e cazul

Cum credeti ca este alocat B-ul in memorie?



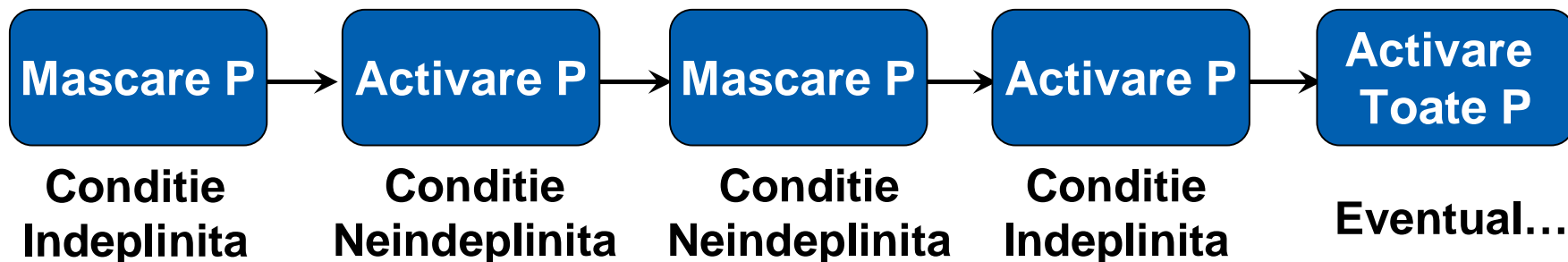
- Structura SIMD
- Setul de Instructiuni Masina SIMD
- Implementarea Salturilor Conditionate in Structura SIMD
- Organizarea Datelor in Structura SIMD



# Implementarea Salturilor Conditionate

12

- In sistemele SIMD salturile conditionate se implementeaza prin divizarea fluxului de instructiuni (doar 1FI posibil!)
  - Un flux pentru conditia indeplinita
  - Alt flux pentru conditia neindeplinita
  - Cele doua FI se executa la momente diferite de timp si secvential! → paralelism  $n/2$
  - Pentru implementarea acestor salturi conditionate se vor utiliza operatii de mascare a activitatii procesoarelor  $P_i$ :





# Mascare/Activare Procesoare

13

- Se considera cate un bit de mascare asociat fiecarui procesor  $P_i$ , intr-un registru de masca (n biti):  $P_i \leftrightarrow \text{MASK}[i]$ :
  - $\text{MASK}[i] = 1 \rightarrow$  procesorul  $P_i$  functioneaza
  - $\text{MASK}[i] = 0 \rightarrow$  procesorul  $P_i$  in asteptare & nu executa operatia curenta
- Operatiile de testare a mastilor se fac in fiecare  $P$  si rezultatele  $\rightarrow$  Reg Index din UCmd  $\rightarrow$  necesare instructiuni logice & de acces la registre Index:
  - Pt stabilirea de conditii
  - Pt implementarea salturilor
  - Lucru cu registre de masti (mascarea efectiva)



# Instructiuni cu Registre de Masti

14

- Compara vector:
  - CLSS A, i ;if ACC[k] < A[0,k] then  
; INDEX[i].bit<sub>k</sub> ← 1
  - CEQL A, i ;else INDEX[i].bit<sub>k</sub> ← 0  
;similar doar cu conditia '=' si nu '<'
- Ramificatie fortata:
  - BRALL i, Adr ;Salt la Adr numai daca INDEX[i] = 1
  - BRNON i, Adr ;Salt la Adr numai daca INDEX[i] = 0
- Operatii logice:
  - AND/OR/XOR i, j ;INDEX[i] ← INDEX[i] op INDEX[j]
- Complementare index:
  - CMP i ;INDEX[i] ← NOT INDEX[i]



## Instructioni cu Registre de Masti (cont)

15

- Incarca/Memoreaza masca de la index:
  - LDMSK i                                   ;MASK  $\leftarrow$  INDEX[i]
  - STMSK i                                   ;INDEX[i]  $\leftarrow$  MASK
- Mentionarea/extragerea primului bit (prioritatea):
  - FRST i                                   ;if INDEX[i]  $\neq$  0 then  
  ;       ramane nemodificat  
  ;else   primul bit 1 ramane asa & restul  
  ;       de biti 1 se trec pe 0
  - FRST se foloseste cand o parte din Procs indeplinesc o conditie si trebuiesc activate
  - NXFIR i                                   ;INDEX[i] = indicele primului bit 1  
  ;if INDEX[i] = 0 then INDEX[i] = 1 (FFF...)
  - NXFIR se utilizeaza in colaborare cu BCAST



## Exemplu - Normalizarea unei Matrice

16

- Normalizarea prin inlocuirea  $A[i,j] \leftarrow A[i,j]/A[0,j]$  cu  $A[0,j] \neq 0$ :

```
for i = 1 to n - 1
    if (A[0,j] != 0) then  $A_{ij} /= A_{oj}$  ( $0 \leq j \leq n - 1$ )
end i loop
```

- Fie  $M[j] = (A[0,j] \neq 0)$  in corespondenta cu mastile  $P_i$ :

```
for i = 1 to n - 1
     $A_{ij} /= A_{oj}, M[j]$ 
end i loop
```

ENXC i, 1	;INDEX[i] = 1 (contor)
SUB ACC	;resetezi toate ACC-urile (XOR AX, AX)
CEQL A, M	;if ACC[k] = A[0,k] then bit k din INDEX[M] = 1
	; else bit k din INDEX[M] = 0
CMP M	;INDEX[M] $\leftarrow$ NOT INDEX[M] (masca ok?)
LDMSK M	;incarcam masca & activam procs
LDNX n, adrN	;la adrN e valoarea n
adr: LOAD A[i]	;ACC[k] $\leftarrow$ A[INDEX[i],k]
DIV A	;doar pe procs unde MASK = 1
STO A[i]	;A[i] $\leftarrow$ ACC[k] (depunere rezultat)
INC i, 1	;INDEX[i]++
CPNX i, n, adr	;salt la "adr" (sfarsitul ciclului)





- Structura SIMD
- Setul de Instructiuni Masina SIMD
- Implementarea Salturilor Conditionate in Structura SIMD
- Organizarea Datelor in Structura SIMD



# Organizarea Datelor in Structura SIMD

18

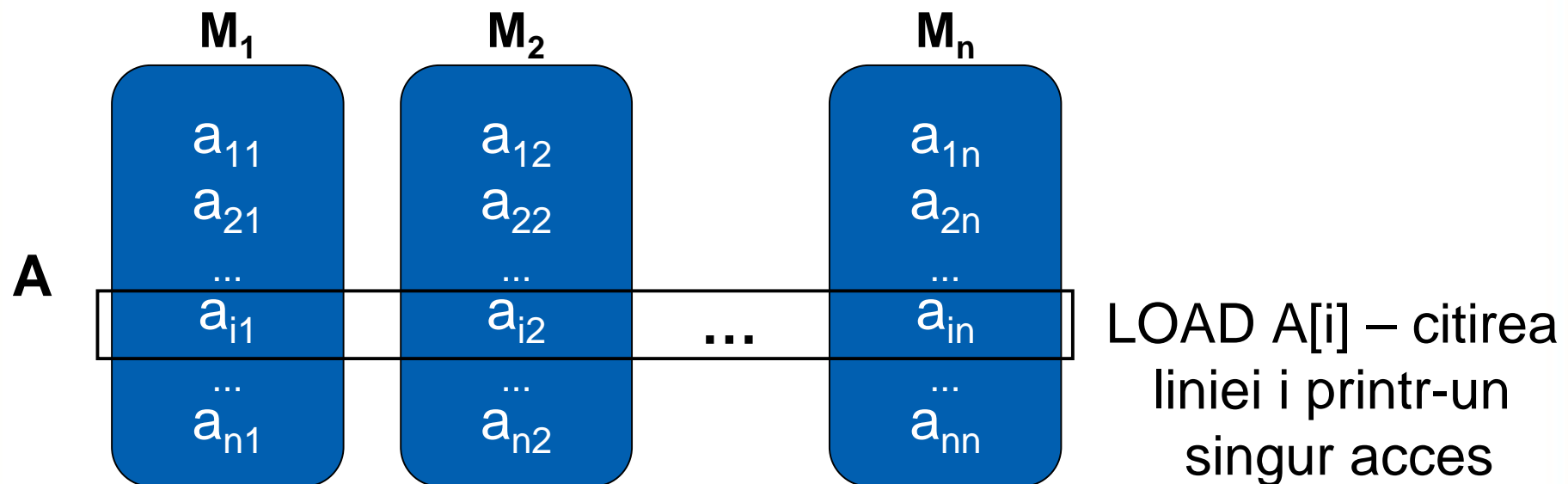
- Probleme:
  - Repartizarea in memoriile  $M_i$  & accesul paralel
  - Citirea vectorilor intr-o **singura instructiune**
  - La matrice: acces la linii si la coloane → necesara transpunerea/deplasarea datelor?
- Memoria:
  - Poate face cel mult un acces la un ciclu de lucru
  - $n$  operanzi in  $n$  module diferite pot fi cititi simultan
  - Defavorabila plasarea in acelasi modul al operanzilor → citirea va fi secventiala in  $n$  cicli de acces!



# Reprezentarea Directa pe Linii

19

- Matrice  $A[n,n]$  & structura SIMD (n procs/memorii)
- Cu acces simultan la liniile matricei:
  - Acces paralel la linii
  - Acces secvential la coloane

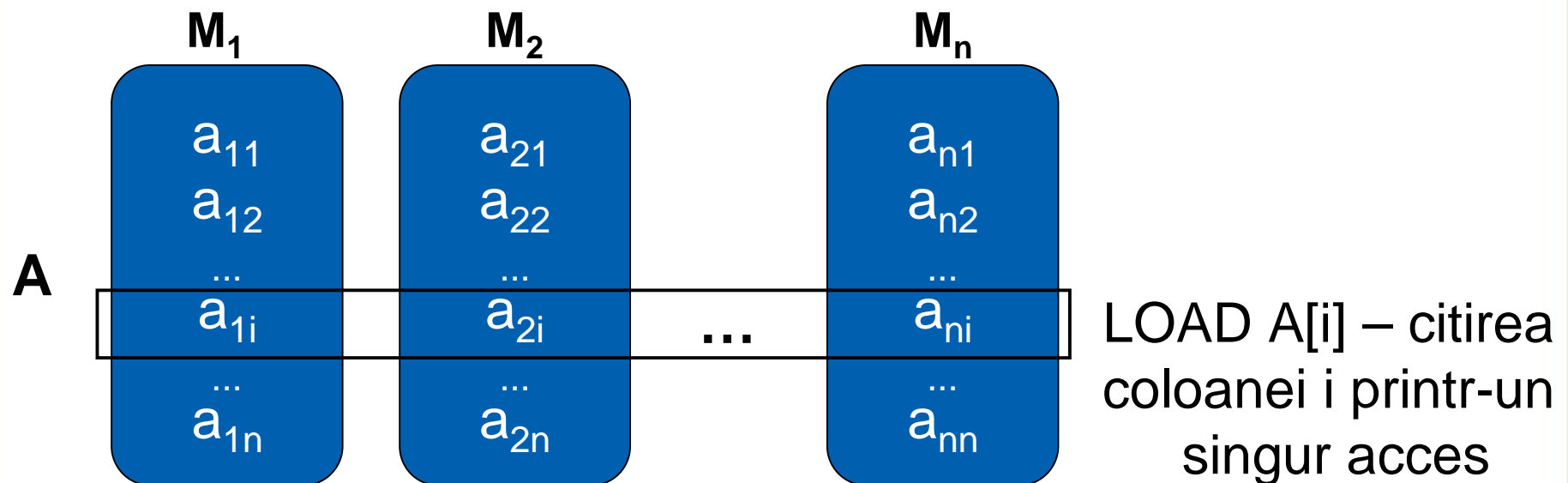




# Reprezentarea Directa pe Coloane

20

- Matrice  $A[n,n]$  & structura SIMD (n procs/memorii)
- Cu acces simultan la coloanele matricei:
  - Acces secvential la linii
  - Acces paralel la coloane



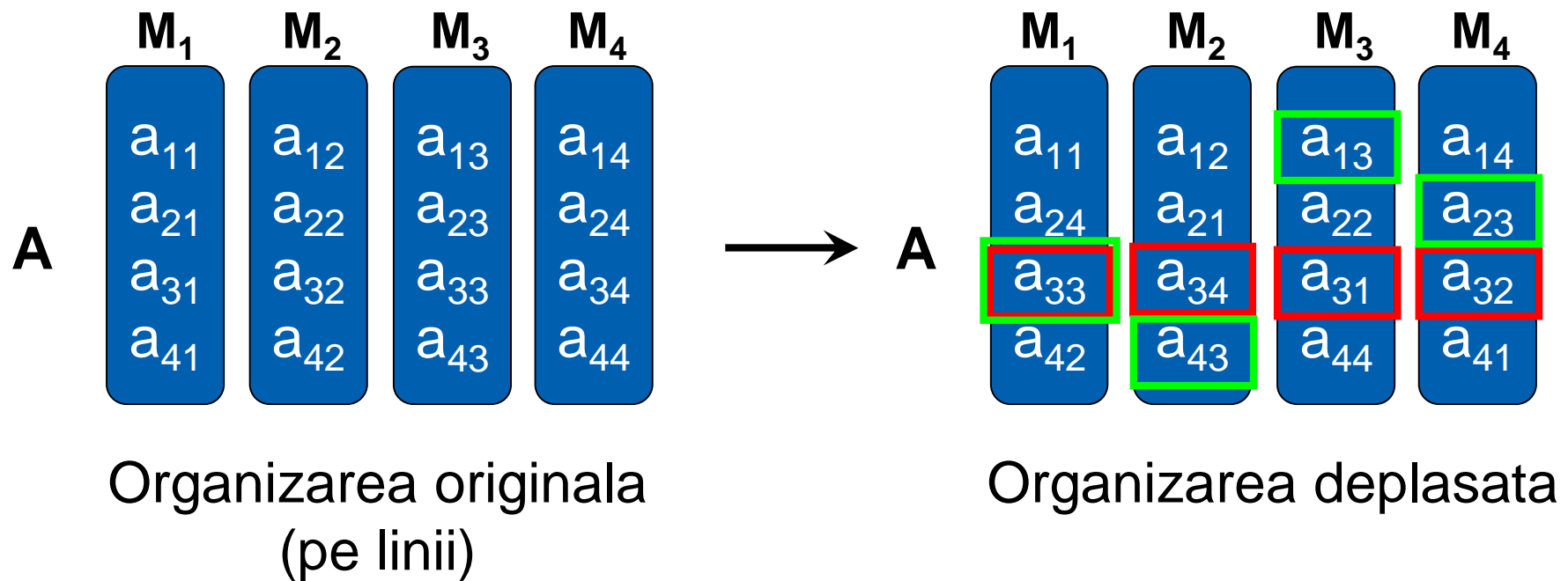
Reprezentarile directe pe linii/coloane sunt incompatibile la același moment de timp



# Reprezentarea Deplasata Ciclica

21

- Matrice  $A[4,4]$  & structura SIMD (4 procs/memorii)
- Cu acces paralel la **liniile** & **coloanele** matricei:
  - Exemplu valid pt inmultirea matricelor (linia & coloana 3)



Necesar un shuffling cu retea de comutatie ( $a_{i1}$  nu e pe  $P_1$ )



# Concluzii Organizarea Datelor

22

- Trade-off simplu:
  - Programe simple → acces secvential la linii & coloane
  - Programe complicate → acces paralel la linii & coloane
- Cu organizarea deplasata accesul la linii & coloane se face intr-un singur ciclu:
  - La linii – adresa fizica/actuala a liniei
  - La coloane – sunt necesari registre index private in  $P_i$ :
    - Pt acces la Col  $i$ , vectorul index e deplasat ciclic pana 0 e la  $M_i$
    - Se difuzeaza adresa liniei 0 cu indicatia de indexare locala
    - Aducerea in pozitia corecta se face prin retea de comutatie P-P
- Costul platit paralelismului este **consistenta datelor!**
- Cu RC cross-bar → nu conteaza asezarea datelor (setare cross-bar sau index local inainte de operatii)



# Exemplu – Suma pe Coloane

23

- Problema:  $A[n, n] \rightarrow$  se dorește suma pe coloane a matricei  $A$  într-un vector  $SUM[n]$
- Cu organizarea pe linii – suma se face secvențial, linie cu linie, fiecare  $P_i$  calculează suma pe câte o coloană

```
 $SUM[k] = 0 \quad (0 \leq k \leq n - 1)$   
for  $i = 0$  to  $n - 1$   
     $LOCINDEX[k] = i \quad (0 \leq k \leq n - 1)$   
     $SUM[k] += A[LOCINDEX[k], k] \quad (0 \leq k \leq n - 1)$   
end i loop
```

- **LOCINDEX** e un index local din  $P$
- Fiecare citire din memorie aduce  $n$  elemente ce sunt adunate la cele  $n$  sume parțiale pe cele  $n$   $P$
- Complexitatea este...  $O(n)$



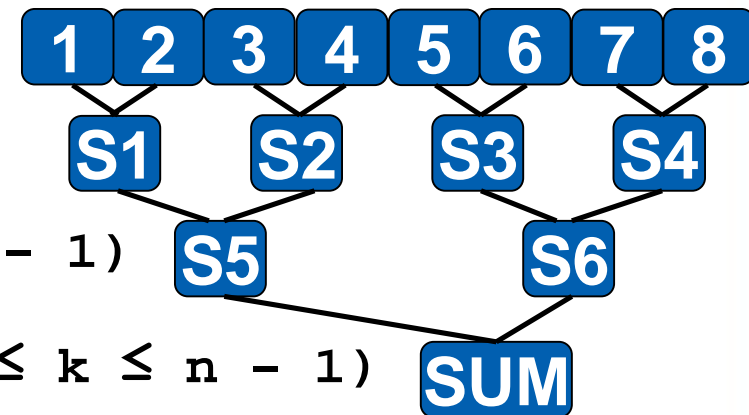
# Exemplu – Suma pe Linii

24

- Problema:  $A[n, n] \rightarrow$  se doreste suma pe linii intr-un vector **SUM**[n]
  1. Organizarea pe linii duce la o complexitate  $O(n^2)$
  2. Organizarea pe coloane duce la un algoritm de transpunere
  3. Folosind registre temporare asociate  $P_i$ -urilor (**TEMP**[k])  $O(n \log_2 n)$
  4. Se doreste o complexitate  $O(n)$  astfel incat trebuie cautata alta solutie

- Solutia 3 este:

```
for i = 0 to n - 1
  TEMP[k] = A[i,k] (0 ≤ k ≤ n - 1)
  for j = 1 to n - 1
    TEMP[k] += TEMP[k-j] (j ≤ k ≤ n - 1)
  end j loop
  SUM[i] = TEMP[n-1]
end i loop
```



$O(n \log_2 n)$





## Exemplu – Suma pe Linii (cont)

25

- Solutia 4: pentru o complexitate  $O(n)$  → se va folosi cate un index local pentru fiecare  $P_i$  (e absolut necesar)
- Algoritmul este:  
`SUM[k] = 0 (0 ≤ k ≤ n - 1)`  
`LOCINDEX[k] = k (0 ≤ k ≤ n - 1)`  
`for i = 0 to n - 1`  
    `SUM[k] += A[LOCINDEX[k], (k - i) mod n] (0 ≤ k ≤ n-1)`  
`end i loop`
- Se aduna cate o **diagonala** la suma curenta
- Diagonala e deplasata ciclic spre elementul adjacent
- $A[\text{LOCINDEX}[k], k]$  nu este altceva decat coloana  $k$  din matrice!
- TA: Rezolvarea sumelor pe linii & coloane cu organizarii deplasate a datelor

$M_1$	$M_2$	$M_3$	$M_4$
$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$
$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$



- Elaborarea unor algoritmi eficienti necesita
  - Organizarea corespunzatoare a structurilor de date
  - Registre locale ale sistemului de calcul
  - Existenta unei retele de comutatie P-P
- Etapele elaborarii unui program sunt:
  - Descrierea algoritmului
  - Paralelizarea algoritmului
  - Optimizarea paralelizarii curente



# What Next?

27

- Q & A?
- Next time:
  - Structuri SIMD:
    - Probleme de Comunicatii intre Procesoarele unui Sistem SIMD
    - Deplasarea Ciclica a Datelor intre Procesoare
    - Intercalarea Perfecta
    - Conectarea Inversa
    - Permutari Elementare