Vektorski procesori

- vektorizacija ugnježđenih petlji
- organizacija memorije i smeštanje podataka

vektorizacija ugnježđjenih petlji (nast.)

- Razmotrimo jedno ovakvo gnezdo petlji

for $I_1 = l_1, u_1$
for $I_2 = l_2, u_2$
:
for $I_n = l_n, u_n$
$S_1(I)$
$S_2(I)$
:
$S_k(I)$
» **endfor**

- gde su l_j, u_j granice petlje:
  - to mogu biti celobrojni izrazi u kojima figurišu indeksi $I_1, I_2, \ldots, I_{j-1}$.
  - I je uredjena n-torka indeksa $(I_1, \ldots, I_n)$.
  - $S_1, \ldots, S_k$, su naredbe dodeljivanja u kojima se pojavljuju indeksirane promenljive

vektorizacija ugnježdjenih petlji (nast.)

- Da bi uočili kakve zavisnosti postoje izmedju naredbi unutar tela petlje potrebno je prvo uočiti sve parove generisanih—korišćenih promenljivih i za svaki takav par odrediti vektor zavisnosti d.
- Od svih vektora zavisnosti formira se matrica zavisnosti po podacima, D
  - Ako je generisana promenljiva X(f(I)), gde je f celobrojna funkcija definisana nad indeksnim skupom I, a X(g(I)) korišćena promenljiva (g je opet celobrojna funkcija definisana nad indeksnim skupom I), vektor zavisnosti se izračunava kao

$$d=f(I) - g(I).$$

Ako su d_1, d_2, ... , d_k svi vektori zavisnosti, tada se matrica zavisnosti po podacima dobija kao

$$D = [\ d_1 \ \ d_2 \ \dots \ d_k]$$

PRIMER

- Pronaći sve vektore zavisnosti u sledećem gnezdu petlji

```
    for i = 1, 5
    for j = 1, 10
    fot k =1, 20
  A(i, j, k) = A(i-1, j, k+1)+ B(i, j, k)
    B(i, j, k+1) = B(i, j-1, k-1) * 3
  endfor{i,j,k}
```

$$D = \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} = \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 2 \end{bmatrix} \end{array} \begin{array}{c} i \\ j \\ k \end{array}$$

matrica zavisnosti

- REŠENJE:
  - Uočimo prvo sve parove generisanih—korišćenih promenljivih:

  A(i, j, k) i A(i-1, j, k+1)

  B(i, j, k+1) i B(i, j, k),

  B(i, j, k+1) i B(i, j-1, k-1)

Odgovarajući vektori zavisnosti

$d_1 = (i, j, k)^T - (i-1, j, k+1)^T = [\ 1, 0, -1]^T$

$d_2 = (i, j, k+1)^T - (i, j, k)^T = [\ 0, 0, 1]^T$

$d_3 = (i, j, k+1)^T - (i, j-1, k-1)^T = [0, 1, 2]^T$

- Prvi element u vektoru koji je ≠0 nosi zavisnost!

vektorizacija ugnježđjenih petlji (nast.)

- Često je za analizu zavisnosti po podacima dovoljno poznavati samo pravce zavisnosti, a ne stvarne vrednosti vektora zavisnosti.
- Pravac zavisnosti se definiše na sledeći način:

$$pravac\_zavisnosti = \begin{cases} <, & if \quad d_i > 0 \\ =, & if \quad d_i = 0 \\ >, & if \quad d_i < 0 \end{cases}$$

- Za prethodne vektore zavisnosti odgovarajući pravci zavisnosti su
  - $d_1 = [\ 1, 0, -1]^T \quad \Rightarrow \quad [\ <, =, >\ ]^T$
  - $d_2 = [\ 0, 0, 1]^T \quad \Rightarrow \quad [\ =, =, <\ ]^T$
  - $d_3 = [0, 1, 2]^T \quad \Rightarrow \quad [\ =, <, <\ ]^T$

$$D = \begin{array}{cc} \begin{bmatrix} < & = & = \\ = & = & < \\ > & < & < \end{bmatrix} & \begin{array}{c} i \\ j \\ k \end{array} \end{array}$$

Pravilo

- Zavisnost nosi prvi element vektora koji je < ili > .
- Pravac = ne sprečava vektorizaciju.

$$D = \begin{array}{cc} \begin{bmatrix} < & = & = \\ = & = & < \\ > & < & < \end{bmatrix} & \begin{matrix} i \\ j \\ k \end{matrix} \end{array}$$

- vektorizacija po indeksnoj promenljivoj *k* nije moguća jer za drugi vektor zavisnosti postoji looop-carry po indeksnoj promenljivoj k

Vektorizacija ugnj. petlji (nast.)

- Da zaista postoji loop-carry zavisnost koja sprečava vektorizaciju po indeksnoj promenljivoj k, možemo se uveriti ako izvršimo odmotavanje petlje po *k* za neku fiksnu vrednost promenljivih *i* i *j*.

❋ na primer, za i=1 i j=1 i k=1, 2,....,20 imamo

- i=1, j=1, k=1 A(1, 1, 1) = A(0, 1, 2) + B(1, 1, 1)
- B(1, 1, 2) = B(1, 0, 0) *3
- loop-carry zavisnost
- i=1, j=1, k=2 A(1, 1, 2) = A(0, 1, 3) + B(1, 1, 2)
- B(1, 1, 3) = B(1, 0, 1) *3

loop-carry zavisnost

- i=1, j=1, k=3 A(1, 1, 3) = A(0, 1, 4) + B(1, 1, 3)
- B(1, 1, 4) = B(1, 0, 2) * 3

Vektorizacija ugnj. petlji (nast.)

- Ako vektorizacija nije moguća po unutrašnjoj petlji, može se pokušati sa zamenom petlji ili nekim drugim transformacijama indeksnih promenljivih.
- Postoje tri elementarne transformacije koje se mogu obavljati nad indeksnim skupovima, tj. nad petljama.
  - Ove transformacije se opisuju pomoću matrica transformacija, T.
- Ove matrice moraju da poseduju sledeće tri osobine:
  1. To su kvadratne matrice, što znači da vrše preslikavanje n-dimenzionalnog indeksnog prostora u n-dimenzionalni indeksni prostor
  2. To su celobrojne matrice
  3. $|\det T| = 1$

Vektorizacija ugnj. petlji (nast.)

- Zbog ovih osobina proizvod dve elementarne transformacije daje važeću transformaciju.
- Da bi jedna transformacija mogla da se primeni nad indeksnim skupom a da to ne utiče na korektnost izračunavanja, matrica transformacije T ne sme da menja znak vektora zavisnosti:
  - Ako je *d* vektor zavisnosti, T matrica transformacije, tada novi vektor zavisnosti $\hat{d}$, koji se dobija kada se T primeni na *d*, tj.

$$\hat{d} = T \cdot d$$

  - mora biti istog znaka kao i *d*.
    - Ako je d>0, tada mora i $\hat{d}$ >0, ili ako je d < 0, tada i $\hat{d}$ mora <0.
    - Za vektor se kaže da je pozitivan (negativan) ako mu je prvi ne-nulti element pozitivan (negativan).

Vektorizacija ugnj. petlji (nast.)

- Vektor
- $$d = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$ je > 0
- $$d = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}$$ je < 0

Elementarne transformacije nad indeksnim skupovima

1. *permutacija*
2. *obrtanje redosleda*
3. *krivljenje*

Permutacija – omogućava zamenu mesta dvema petljama

$$I_1 = l_1, u_1$$
$$I_2 = l_2, u_2$$
$$\vdots$$
$$I_m = l_m, u_m$$

Ako želimo da zamenimo mesta petljama po indeksima I_j i I_k, ta transformacija se opisuje pomoću matrice T koja se dobija kada se u jediničnoj matrici zamene mesta j-toj i k-toj vrsti.

Transformacija permutacije - primer

- za m=2 matrica transformacije T je oblika

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Ako smo imali petlje

I = 1, n

J = 1, m

nakon transformacije T dobićemo

J= 1, m

I = 1, n

Transformacija permutacije

- PRIMER: Izvršiti permutaciju indeksnih promenljivih I i J u sledećem gnezdu petlji

```
for I = 1, n
  for J = 1, n
 A(J) = A(J) + C(I, J)
endfor{I,J}
```

- Primenom transformacije permutacije nad indeksnim skupom $(I, J)^T$ dobijamo nove indeksne promenljive U i V na sledeći način

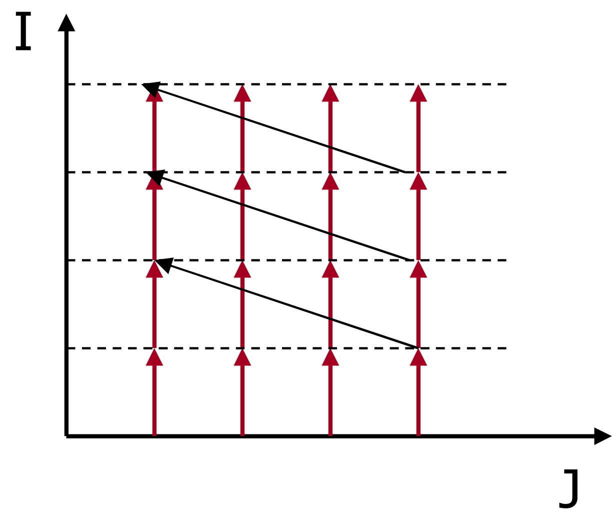
$$\begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} I \\ J \end{bmatrix} = \begin{bmatrix} J \\ I \end{bmatrix}$$

- pri čemu je U = J, a V = I.
- Granice za U i V odredjujemo na osnovu granica za indeksne promenljive J i I, respektivno.
- Transformisano gnezo petlji sada ima oblik

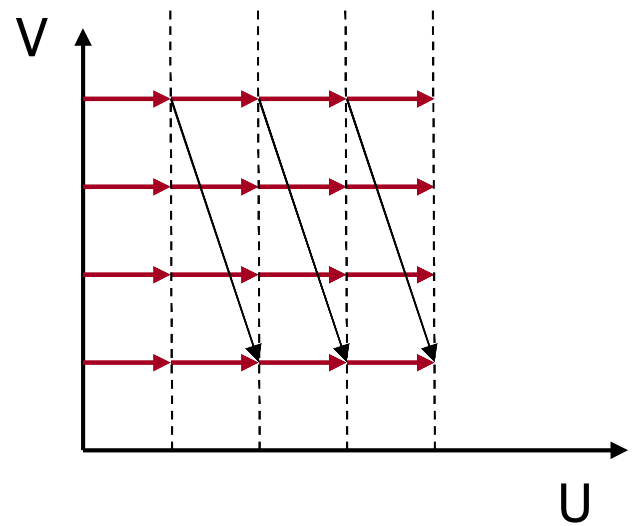
```
for 10  U = 1, n
  for 10  V = 1, n
 A(U) = A(U) + C(V, U)
endfor{u,v}
```

Transformacija permutacije

- Transformacijom je promenjen redosled izračunavanja elemenata vektora A.



Pre transformacije



Posle transformacije

Transformacija obrtanje

- Omogućava promenu redosleda izračunavanja po odredjenoj indeksnoj promenljivoj
- Transformacija se opisuje jediničnom matricom u kojoj je u i-toj vrsti dijagonalni element jednak –1.

- PRIMER: Posmatrajmo ovakvo gnezdo petlji:

```
for i = 1, n
for j = 1, n
  A(i, j) = A(i-1, j+1)*k
 endfor{i,j}
```

- Želimo da primenimo transformaciju obrtanja po indeksnoj promenljivoj *j*.
- Matrica transformacije je oblika

$$T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Obrtanje – primer (nast.)

- Novi indeksni skup dobijamo na sledeći način

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ -j \end{bmatrix}$$

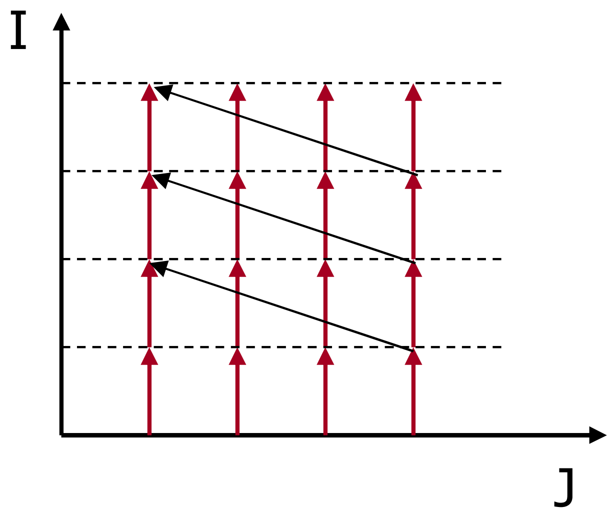
- Što znači da je $u = i$, $v = -j$, a granice indeksa u i v su u=1, n i v=-n, -1.

- Transformisana petlja ima sledeći izgled

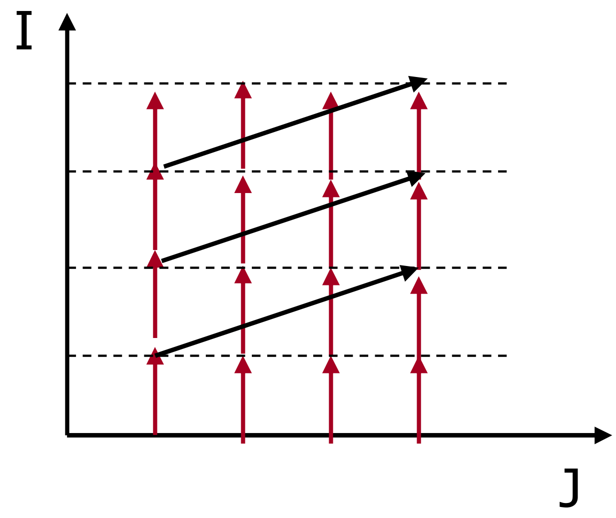
```
for u = 1, n
for v = - n, -1
A(u, -v) = A(u-1, -v+1)*k
endfor{u,v}
```

- Transformacijom je izmenjen redosled izračunavanja po indeksnoj promenljivoj *j*

Obrtanje – primer



Pre transformacije



Posle transformacije

- Npr. za n=3 i i=1 pre transformacije se redom izračunavaju elementi
  A(1,1), A(1,2) i A(1,3),
- Nakon transformacije redosled izračunavanja elemenata je

  A(1,3), A(1,2) i A(1,1).

3. Transformacija krivljenja (skewing)

- Ovom transformacijom se obavlja krivljenje (skeweing) jednog iterativnog indeksa u odnosu na drugi za faktor **f**.
- Pretpostavimo da imamo ovakvu iteraciju indeksnih promenljivih

$$(p_1, p_2, \ldots, p_i, \ldots, p_{j-1}, p_j, p_{j+1}, \ldots, p_n)$$

- ako primenimo krivljenje petlje I_j u odnosu na I_i za faktor f izvršiće se preslikavanje gornje iteracije u

$$(p_1, p_2, \ldots, p_i, \ldots, p_{j-1}, p_j + f \cdot p_i, p_{j+1}, \ldots, p_n)$$

Krivljenje

- Krivljenje petlje i_k u odnosu na i_j za faktor f

$$
\begin{bmatrix}
1 & 0 & .. & & & & .. & 0 & 0 \\
0 & 1 & .. & & & & .. & 0 & 0 \\
\vdots & & & & & & & \vdots & \vdots \\
0 & 0 & .. & 1 & .. & & .. & 0 & 0 \\
\vdots & & & & & & & \vdots & \vdots \\
0 & 0 & .. & f & .. & 1 & .. & 0 & 0 \\
\vdots & & & \vdots & & & .. & \vdots & \vdots \\
0 & 0 & .. & & & & .. & 1 & 0 \\
0 & 0 & .. & & & & .. & 0 & 1
\end{bmatrix}
\begin{bmatrix}
i_1 \\ i_2 \\ \\ ij \\ \\ i_k \\ \\ \\ i_n
\end{bmatrix}
=
\begin{bmatrix}
i_1 \\ i_2 \\ \\ ij \\ \\ i_k + fij \\ \\ \\ i_n
\end{bmatrix}
$$

Krivljenje – primer

- PRIMER: Posmatrajmo sledeće gnezdo petlji

```
for i = 1, n
for j = 1, n
  A(i, j) = A(i, j-1) + A(i-1,j)
endfor{i,j}
```

- Primenimo krivljenje indeksne promenljive *j* u odnosu na *i* za faktor 2.
- Transformacija se opisuje na sledeći način $T = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$
- Novi indeksni skup je $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ j+2i \end{bmatrix}$
- što znači da je $u=i$, $v=2i+j$,
- granice novih indeksnih promenljivih su $u=1,n$ i $v=2u+1,2u+n$.

⁕ Matrica krivljenja

$$T = \begin{bmatrix} 1 & 0 \\ q & 1 \end{bmatrix}$$

Original Loop

```
do I1 = n1, N1, 1
  do I2 = n2, N2, 1
     H(I1, I2)
  end do
end do
```

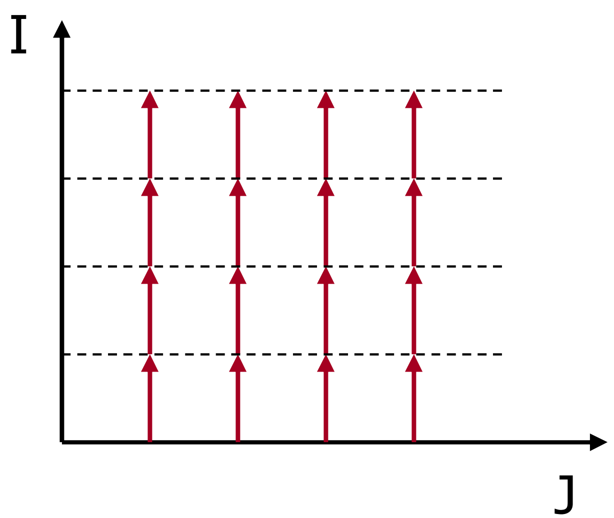
Transformed Loop

```
do K 1 = n1, N1, 1
  do K 2 = n2 + q*K 1, N2 + q*K 1, 1
          H(K 1, K 2 - q*K 1)
  end do
end do
```

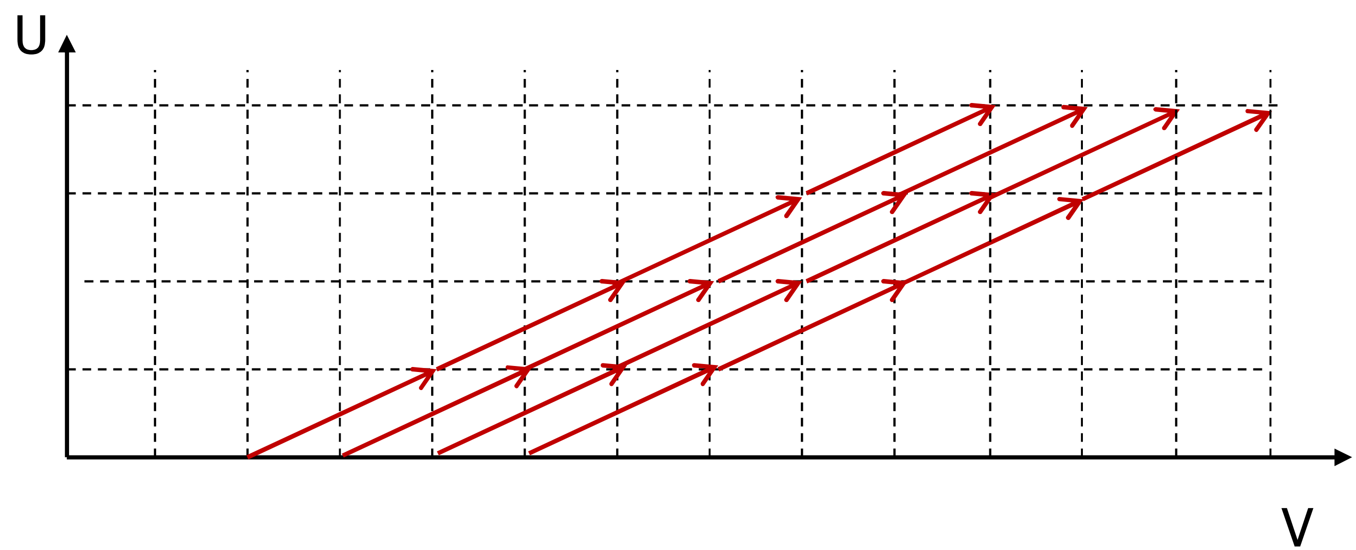
Krivljenje – primer (nast.)

- Transformisana petlja ima sledeći izgled

  for u = 1, n
  for v = 2u+1, 2u+ n
  A(u, v-2u) = A(u,v-2u-1) + A(u-1, v-2u)
  endfor{u,v}



Pre transformacije



Posle transformacije

Primer-1

- Da bi neka transformacija mogla da se primeni nad indeksnim skupom ona ne sme da menja znak vektora zavisnosti da bi se sačuvale zavisnosti koje postoje u redosledu izračunavanja

```
    for  i = 1, 5
    for  j = 1, 10
    for  k =1, 20
A(i, j, k) = A(i-1, j, k+1)+ B(i, j, k)
    B(i, j, k+1) = B(i, j-1, k-1) * 3
endfor{i,j,k}
```

$$D = \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} = \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 2 \end{bmatrix} \end{array} \begin{array}{c} i \\ j \\ k \end{array}$$

matrica zavisnosti

- REŠENJE:
  - Uočimo prvo sve parove generisanih—korišćenih promenljivih:

A(i, j, k) i A(i-1, j, k+1)

B(i, j, k+1) i B(i, j, k),

B(i, j, k+1) i B(i, j-1, k-1)

Odgovarajući vektori zavisnosti

$d_1 = (i, j, k)^T – (i-1, j, k+1)^T = [\ 1, 0, -1]^T$

$d_2 = (i, j, k+1)^T – (i, j, k)^T = [\ 0, 0, 1]^T$

$d_3 = (i, j, k+1)^T – (i, j-1, k-1)^T = [0, 1, 2]^T$

- Zbog vektora d2 nije moguće izvršiti vektorizaciju.
  - permutacijom petlji j i k dobićemo kod koji je moguće vektorizovati

Primer-1

- Matrica zavisnosti pre permutacije

$$D = [d_1 \quad d_2 \quad d_3] = \begin{array}{cc} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 2 \end{bmatrix} & \begin{array}{c} i \\ j \\ k \end{array} \end{array}$$

- Matrica zavisnosti nakon permutacije

$$D = [d_1 \quad d_2 \quad d_3] = \begin{array}{cc} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} & \begin{array}{c} i \\ k \\ j \end{array} \end{array}$$

- zavisnosti su zadržane ali se unutrašnja petlja može vektorizovati

Primer-1–nast.

- transformisana petlja

```
for  i = 1, 5
for  k = 1, 20
for  j = 1, 10
A(i, j, k) = A(i-1, j, k+1)+ B(i, j, k)
      B(i, j, k+1) = B(i, j-1, k-1) * 3
endfor{i,j,k}
```

- i=1,k=1,j=1
  - a(1,1,1)=a(0,1,2)+b(1,1,1)
  - b(1,1,2) =b(1,0,0)*3
- i=1,k=1,j=2
  - a(1,2,1)=a(0,2,2)+b(1,2,1)
  - b(1,2,2) =b(1,1,0)*3

⋮

- i=1,k=2,j=1
  - a(1,1,2)=a(0,1,3)+b(1,1,2)
  - b(1,2,3) =b(1,0,1)*3

Primer2

- Da bi neka transformacija mogla da se primeni nad indeksnim skupom ona ne sme da menja znak vektora zavisnosti da bi se sačuvale zavisnosti koje postoje u redosledu izračunavanja.
  - PRIMER: Ako imamo ovakvo jedno gnezdo petlji
  - for i = 1, 3
  - for j = 1, 2
  - A(i, j) = A(i-1, j+1) *2
  - endfor{i,j}
  - vektor zavisnosti je $d=(i, j)^T - (i-1, j+1)^T = (1, -1)^T > 0$.
  - Ako bismo primenili transformaciju permutacije, narušili bismo zavisnosti koje postoje u redosledu izračunavanja jer je

$$\hat{d} = T \cdot d = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} < 0$$

  - a vektor d je >0!

Primer2 – nast.

```
for i = 1, 3
for j = 1, 2
A(i, j) = A(i-1, j+1) *2
endfor{i,j}
```

za i=1, j=1 izračunava se A(1,1)=A(0,2)*2
j=2 izračunava se A(1,2)=A(0,3)*2

za i=2, j=1 izračunava se A(2,1)=A(1,2)*2
j=2 izračunava se A(2,2)=A(1,3)*2

za i=3, j=1 izračunava se A(3,1)=A(2,2)*2
j=2 izračunava se A(3,2)=A(2,3)*2

- Strelicama je označen redosled zračunavanja koji mora biti ispoštovan:
  - A(1, 2) mora biti izračunat pre A(2,1) jer A(2,1) koristi vrednost A(1,2).
  - A(2,2) mora biti izračunat pre A(3,1)

- Ako na prethodnu petlju primenimo transformaciju permutacije dobićemo $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} j \\ i \end{bmatrix}$

```
for u = 1, 2
for v = 1, 3
A(v, u) = A(v-1, u+1) *2
enfor{u,v}
```

za u=1, v=1 izračunava se A(1,1)=A(0,2)*2
v=2 izračunava se A(2,1)=A(1, 2)*2
v=3 izračunava se A(3,1)=A(2, 2)*2

za u=2, v=1 izračunava se A(1,2)=A(0,3)*2
v=2 izračunava se A(2,2)=A(1, 3)*2
v=3 izračunava se A(3,2)=A(2, 3)*2

- Prvo izračunava element A(2,1) pa nakon toga element A(1,2), što je pogrešno!

Kompozicija transformacija

- Zbog osobina elementarnih matrica transformacija, proizvod elementarnih matrica transformacija daje takodje validnu transformaciju.
  - Tako, da bi u prethodnom primeru mogli da primenimo permutaciju, a da ne narušimo zavisnosti po podacima, možemo da primenimo kompoziciju transformacija permutacije i obrtanja:

$$T = T_{\text{permutacija}} T_{\text{obrtanje}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

  - Novi vektor zavisnosti biće pozitivan, tj.

$$\hat{d} = T \cdot d = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} > 0$$

Primer (nast.)

for i = 1, 3
for j = 1, 2
A(i, j) = A(i-1, j+1) *2
endfor{i,j}

za i=1, j=1 izračunava se A(1,1)=A(0,2)*2
j=2 izračunava se A(1,2)=A(0,3)*2

za i=2, j=1 izračunava se A(2,1)=A(1,2)*2
j=2 izračunava se A(2,2)=A(1,3)*2

za i=3, j=1 izračunava se A(3,1)=A(2,2)*2
j=2 izračunava se A(3,2)=A(2,3)*2

Kompozicija transformacija permutacija+obrtanje

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} -j \\ i \end{bmatrix}$$

- transformisana petlja

for u=-2,-1
for v=1,3
A(v,-u)=A(v-1,-u+1)*2
endfor{u,v}

za u=-2, v=1 izračunava se A(1,2)=A(0,3)*2
v=2 izračunava se A(2,2)=A(1,3)*2
v=3 izračunava se A(3,2)=A(2,3)*2

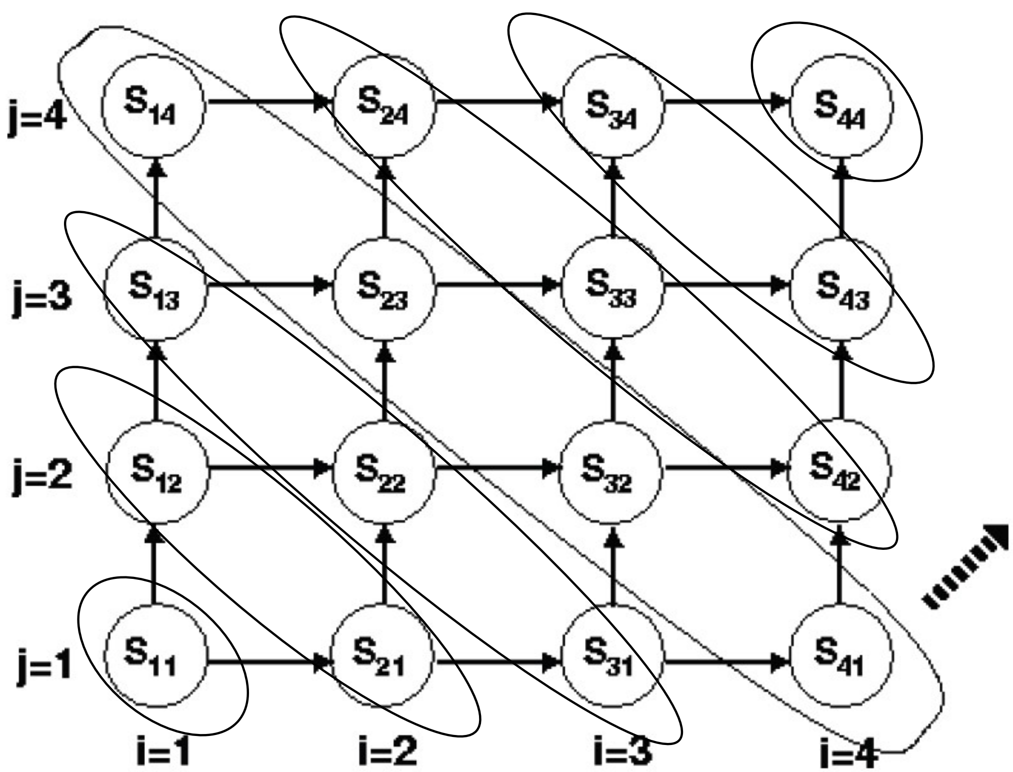
za u=-1, v=1 izračunava se A(1,1)=A(0,2)*2
v=2 izračunava se A(2,1)=A(1,2)*2
v=3 izračunava se A(3,1)=A(2,2)*2

Redosled izračunavanja je ispoštovan!

Primer

- Da li se sledeća petlja može vektorizovati?

```
for i = 1:N
for j = 1:M
A(i,j) = A(i-1,j) + A(i,j-1);
```



Krivljenje?
permutacija?

Vektorski računari

Smeštanje podataka i odredjivanje broja memorijskih banaka

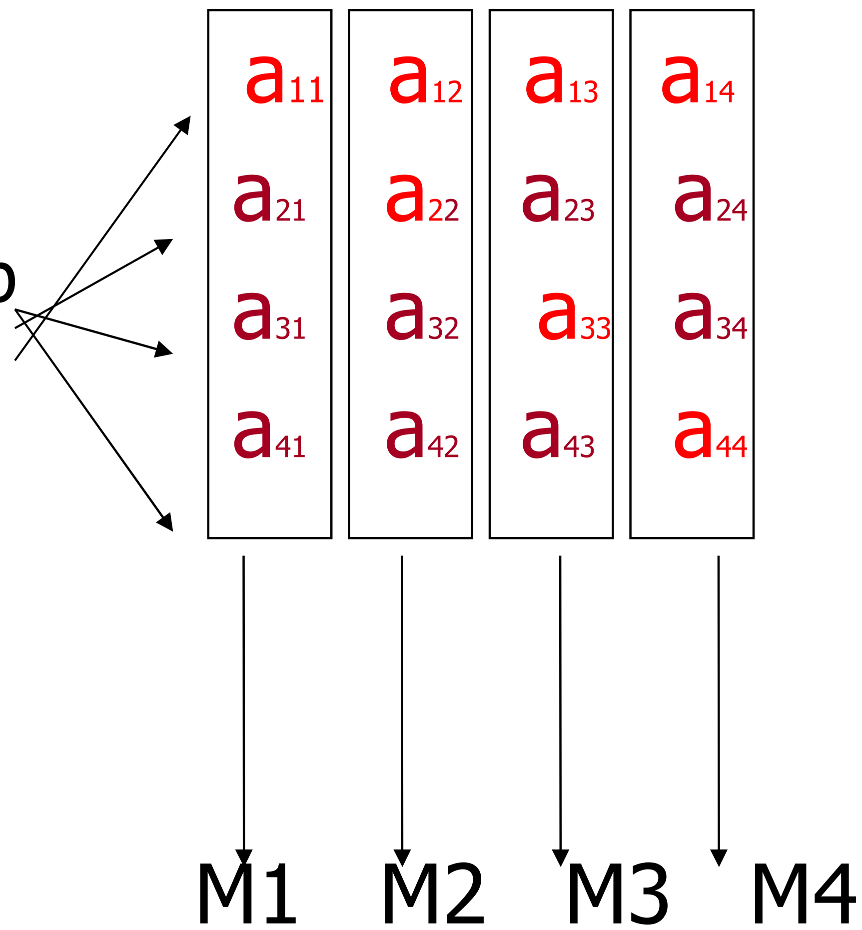
Smeštanje podataka

- Jedan od ključnih faktora koji utiče na vreme izvršenja programa na paralelnom procesoru je latentnost memorijskog sistema
  - vreme od trenutka izdavanja zahteva za pribavljanjem podatka do trenutaka kada on postane dostupan
    - da bi se problem rešio koristi se paralelne memorijske banke
    - način smeštanja podataka igra važnu ulogu, jer može smanjiti vreme pristupa elementima polja
- Pošto je osnovna struktura podataka koja se koristi kod vektorskih računara polje, način smeštanja elemenata polja u memorijske module može bitno uticati na efikasnost vektorskog izračunavanja tako što će smanjiti vreme pristupa elementima polja

Smeštanje podataka – primer

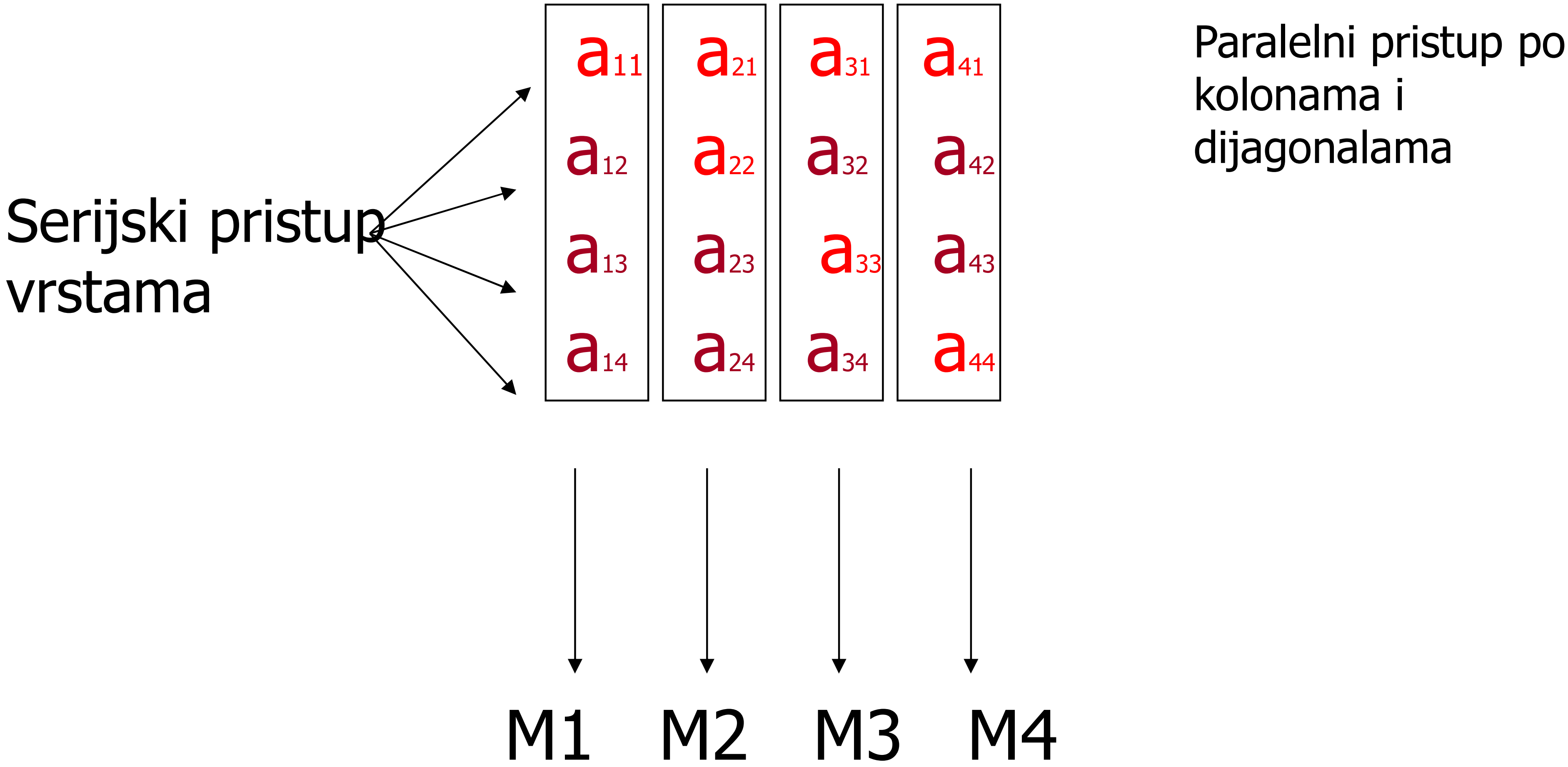
Razmotrimo moguće načine smeštanja elemenata matrice A dimenzija 4x4 u mem. banke,

Serijski pristup kolonama



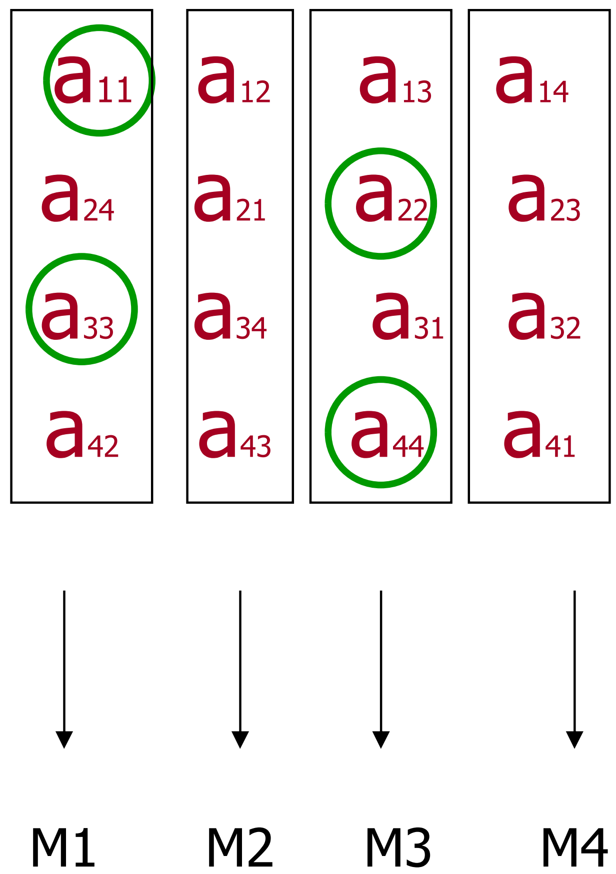
Moguć je paralelni pristup elementima vrsta i dijagonala

Primer - nastavak



Primer - nastavak

Paralelni pristup elementima kolona i vrsta

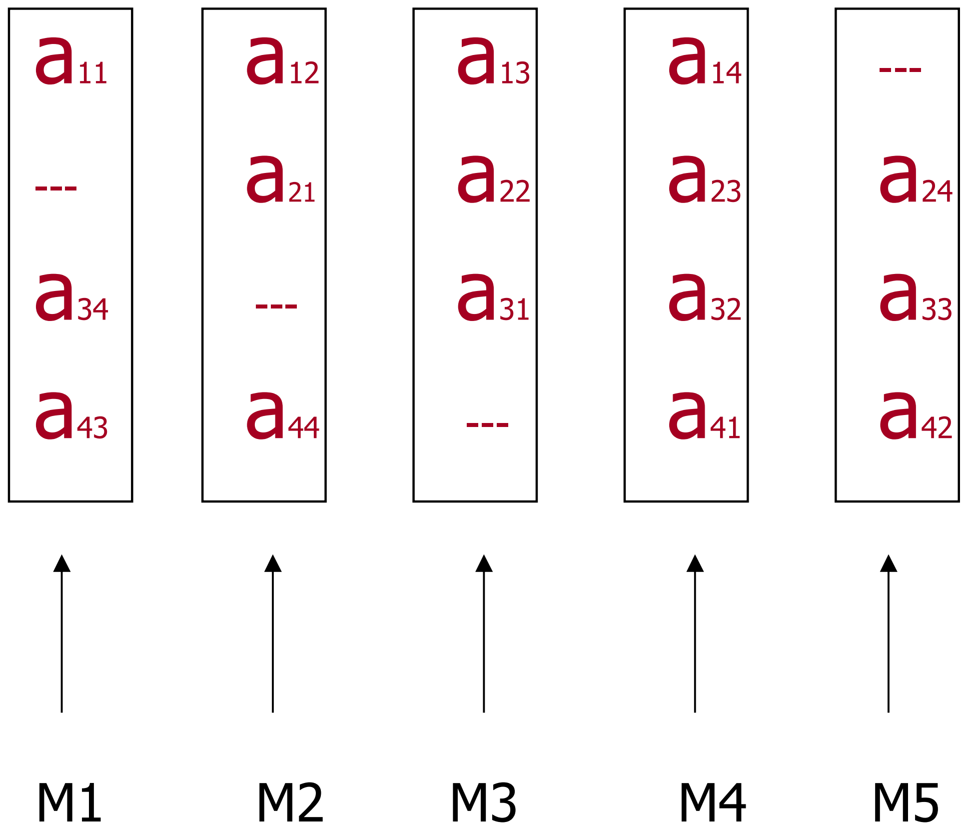


Konflikti kod pristupa dijagonalnim elementima!

Primer - nastavak

Paralelni pristup kolonama/vrstama/dijagonalama

(5 memorijskih banaka)



- Teorijski, da bi se obezbedio pristup proizvoljnoj strukturi dvodimenzionalnog polja, potrebno je da broj memorijskih banaka bude veći od dimenzije matrice

Odredjivanje broja memorijskih banaka

- Primer: množenje matrica

```
for i =1, 100
for j =1, 100
A(i,j) = 0.0
do 10 k =1, 100
A(i,j) = A(i,j) +B(i,k) * C(k,j)
endfor{i,j}
```

- Petlju po indeksnoj promenljivoj k je moguće vektorizovati tako što bi se obavilo množenje vrsta matrice B kolonama matrice C.

```
for i =1, 100
for j =1, 100
A(i,j) = 0.0
A(i,j) = A(i,j) +B(i,1:100) * C(1:100,j)
endfor{i,j}
```

- Da bi ustanovili efikasnost vektorizacije moramo razmotriti kako su susedni elementi u matricama B i C adresirani.

Odredjivanje broja memorijskih banaka (nast.)

- Kada se vrši smeštanje elemenata dvodimenzionalnog polja u memeoriju vrši se linearizacija .
  - Ako se u sukcesivne mem. lokacije smeštaju elementi kolona, znači da u primeru množenja matrica elementi matrice B kojima treba pristupiti u jednoj iteraciji nisu na sukcesvnim memorijskim adresama već su udaljeni medjusobno za

$$broj\_vrsta \; x \; dužina\_reči\_u\_bajtovima$$

  - Razmak izmedju susednih elemenata kojima treba pristupiti sukcesivno zove se vektorski korak ili pomeraj (vector stride).
  - U primeru množenja matrica, vektorski korak za matricu C je 1, dok je za B jednak 100.
  - Kada se vektor pribavi u vektorski registar on se ponaša kao da ima logički susedne elemente.

Odredjivanje broja memorijskih banaka (nast.)

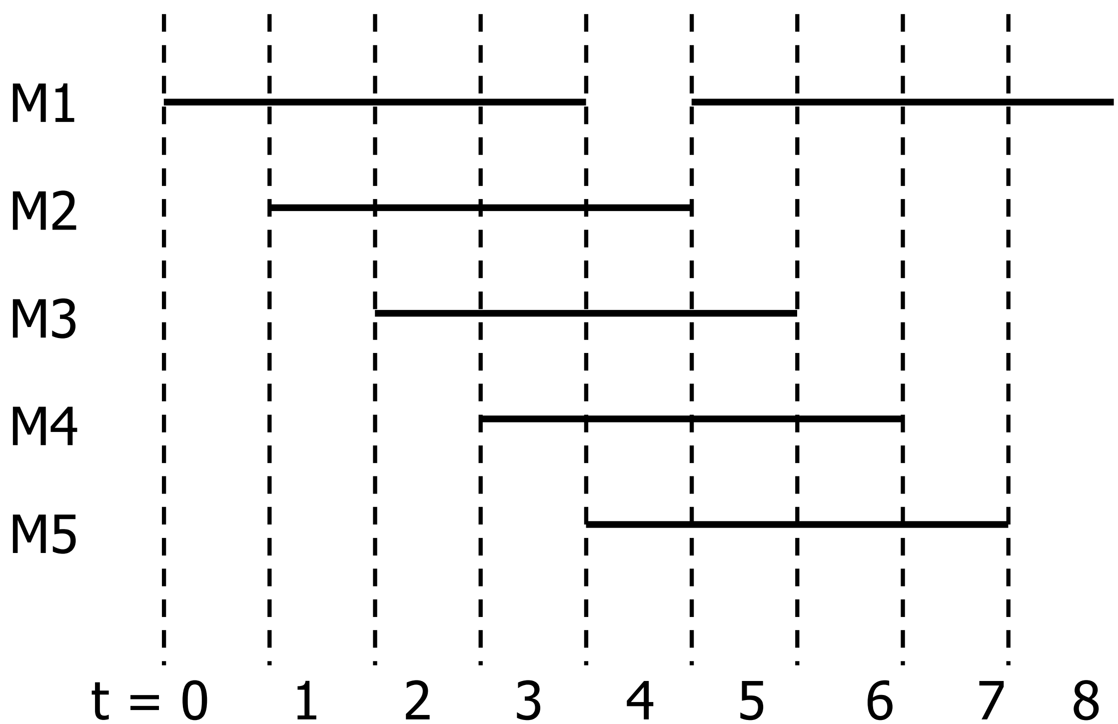
- Vektorski procesor može da upravlja vektorskim korakom > 1 pomoću vektorskih LOAD/ STORE operacija.
  - Ova sposobnost da pristupa nesekvencijalnim memorijskim lokacijama i da ih prevodi u strukturu sa logički susednim elementima je jedna od značajnih prednosti vektorskih procesora nad keš baziranim procesorima.
    - Vektorski korak kao i startna adresa vektora mogu se zapamtiti u neki od registara opšte namene.
    - Zatim se instrukcija tipa LVWS (load_vector_with_stride -- napuni vektor sa korakom) može upotrebiti da se pribavi vektor u vektorski registar (slično i za STORE naredbu SVWS).
    - Kod nekih vektorskih procesora LOAD i STORE uvek imaju vrednost koraka zapamćenu u registru, pa nema posebnih LOAD i STORE instrukcija

Odredjivanje broja memorijskih banaka (nast.)

- Komplikacije u memorijskom sistemu mogu da nastupe kad treba podržati korake > 1.
  - U opštem slučaju da bi se mogao pribaviti jedan element vektora po klok ciklusu, broj memorijskih banaka mora biti veći od latentnosti memorijskog sistema (vreme pristupa memeoriji).
  - Kada postoji korak > 1 može se desiti da se zahteva pristup istoj memorijskoj banci većom brzinom od brzine memorijskog ciklusa.
    - U takvim slučajevima jedan zahtev mora biti zakašnjen zbog postojanja konflikta.

A1	A2	A3	A4	A5
A6	A7	A8	A9	A10
M1	M2	M3	M4	M5

Latentnost =4



Odredjivanje broja memorijskih banaka (nast.)

- Konflikt kod pristupa memorijskoj banci nastupa ako važi sledeći uslov

$$\frac{NZS(vektorski\_korak, Broj\_memorijskih\_banaka)}{vektorski\_korak} \leq latentnost\_memorijskog\_sistema$$

  - NZS je najmanji zajednički sadržalac.
- **PRIMER**. Pretpostavimo da imamo 16 memorijskih banaka sa latentnošću od 12 clk ciklusa. Koliko vremena će biti potrebno da se napuni 64-elementni vektor ako se elementi koji se pribavljaju nalaze na medjusobnom rastojanju
  - 1 (tj. vektorski korak je 1)
  - 32 (tj. vektorski korak je 32)

Odredjivanje broja memorijskih banaka -primer

- **REŠENJE:**
  - a) Pošto je broj memorijskih banaka (16) veći od latentnosti memorijskog sistema (12) za korak 1 imaćemo da je vreme potrebno da se pribave 64 elementa
- 12 + 63 = 75 clk ciklusa ili 1.2 clk/element
  - b) $$\frac{NZS(32,16)}{32} = \frac{32}{32} = 1 < 16$$
  - Najgori mogući vektorski korak je umnožak od broja memeorijskih banaka, kao u ovom primeru.
    - U ovakvim sitacijama svi elementi kojima treba pristupiti se nalaze u istoj memorijskoj banci pa je latentnost memorijskog sistema vidljiva za svaki elelent koji treba pribaviti umesto samo jednom kod prvog pristupa kada je korak 1.
    - U našem primeru to dovodi do latentnosti od 12 clk ciklusa po elementu, tj. za ceo vektor: **12 x 64 = 768 clk**

* Konflikt kod pristupa memoriji neće nastupiti ako su vektorski korak i broj memeorijskih banaka uzajamno prosti brojevi i ako postoji dovoljno memorijskih banaka da ne nastupi konflikt kada je vektorski korak 1 (odnosno ako je broj banaka veći od latentnosti memeorijskog sistema).