

Vektorizacija petlji

- * Da bi se uspešno koristila vektorska mašina potrebni su ili efikasni viši programski jezici ili vektorizirajući kompjajleri .
 - U prvom slučaju deo programa koji se može izvršiti u vektorskem obliku eksplicitno specificira programer korišćenjem odgovarajućih naredbi višeg programskog jezika.
 - U drugom slučaju program je napisan na nekom standardnom sekvencijalnom programskom jeziku, a kompjajler određuje koji deo programa se može vektorizovati.
- * Jedini kandidati za vektorizaciju su petlje.
 - Uvek se vektorizuje najdublje ugnježdjena petlja.
- * Kompajler mora biti u stanju da prepozna li se petlja ili deo petlje može vektorizovati i da shodno tome generiše odgovarajući vektorski kod.
 - Da bi to učinio kompjajler mora utvrditi koje zavisnosti postoje izmedju promenljivih u petlji.
 - Samo RAW hazardi mogu sprečiti vektorizaciju.

Primer

```
for i=1 to n  
S1    A(i+1)=A(i)+B(i)  
S2    B(i+1)=B(i)*A(i+1)  
      endfor
```

* U petlji je moguće uočiti sledeće zavisnosti:

- Naredba S1 koristi vrednost koju je S1 izračunala u prethodnoj iteraciji.
 - Ovakva zavisnost postoji u ovom primeru, jer S1 u i+1-iteraciji koristi vrednost A(i) koja je izračunata u i-toj iteraciji kao A(i+1). Isto važi i za naredbu S2 kada su u pitanju B(i) i B(i+1).
- S1 koristi vrednost koju računa S2 u prethodnoj iteraciji.
 - I ovakav tip zavisnosti postoji u ovom primeru: S1 koristi vrednost B(i) u i+1-iteraciji koja je izračunata kao B(i+1) u i-toj iteraciji
- S2 koristi vrednost izračunatu u S1 u istoj iteraciji,
 - tj. A(i+1)

Zašto do ovih zavisnosti dolazi?

- * Pošto su vektorske operacije protočne i latentnost može biti veoma duga, prethodna iteracija može da se ne završi a da sledeća već otpočne.
 - Tj. $i+1$ -iteracija može otpočeti a da se i -ta iteracija ne završi.
 - Tako se može desiti da se rezultat i -te iteracije ne upiše, a da $i+1$ -iteracija krene.
 - Shodno tome, ako postoji situacija tipa 1 i 2, vektorizacija petlje će dovesti do RAW hazarda (hazarda koji vektorska mašina ne proverava).
 - Zbog toga se ovakve petlje ne mogu vektorizovati

Analiza zavisnosti

- * Zavisnost se može uočiti ako se izvrši razvijanje petlje po indeksnoj promenljivoj:

for i=1 to 100

$$A(i+1)=A(i)+B(i)$$

$$B(i+1)=B(i)*A(i+1)$$

endfor

- za $i=1$

$$A(2)=A(1)+B(1)$$

$$B(2)=B(1)*A(2)$$

- $i=2$

$$A(3)=A(2)+B(2)$$

$$B(3)=B(2)*A(3)$$

- Crvene strelice prikazuju zavisnosti izmedju iteracija.

klasičan
RAW

- * U situaciji 3 noramlni hardver za detekciju zavisnosti može otkriti ovaj hazard i zaustaviti izdavanje vektorske instrukcije.

- Zbog toga petlje koje sadrže samo ovakve zavisnosti mogu biti vektorizovane.
- *Zavisnosti koje su posledica korišćenja vrednosti izvračunatih u nekoj od prethodnih iteracija zovu se **loop-carry** (prenos iz petlje) zavisnosti*
- Prvi zadatak kompjlera je da utvrdi da li u petlji postoje loop-carry zavisnosti.
- Kompajler to ostvaruje pomoću algoritma za analizu zavisnosti
- analiza može biti veoma kompleksna.

Analiza zavisnosti

- * Najjednostavniji slučaj nastupa kada se ime polja nalazi samo sa jedne strane naredbe dodeljivanja

for i=1 to 100

A(i)=B(i)+C(i)

D(i)=A(i)*E(i)

endfor

- * U ovom slučaju ne može doći do loop-carry zavisnosti.

- Treći tip zavisnosti postoji, ali ovaj tip zavisnosti procesor može detektovati i zaustaviti izdavanje naredbe za množenje, mada bi $+ i *$ mogle paralelno da se izvršavaju jer ne koriste iste funkcionalne jedinice.

Analiza zavisnosti

- * Često se isto ime pojavljuje i kao izvor i kao odredište unutar petlje:

```
for 10 i=1 to 100
y(i)=ax(i)+y(i)
endfor
```
- * U ovom slučaju ne postoji loop-carry zavisnost jer rezultat izračunavanja y ne zavisi od prethodnog izračunavanja.
 - Ovaj problem se može rešiti preimenovanjem promenljivih

Analiza zavisnosti

- * Sledeća petlja, koja se često zove rekurentna petlja, sadrži loop-carry zavisnost:

for i=2 to 100

$$y(i) = y(i-1) + y(i)$$

endfor

- * Zavisnost se može uočiti razvijanjem petlje:

za $i=2$

$$y(2) = y(1) + y(2)$$



za $i=3$

$$y(3) = y(2) + y(3)$$

Kako kompjuter detektuje loop carry zavisnosti generalno?

for $i_1=L_1, U_1$

for $i_2=L_2, U_2$

for $I_n=L_n, U_n$

S1: $A(f(i_1, i_2, \dots, i_n)) = \dots$

S2: $\dots = A(g(i_1, i_2, \dots, i_n))$

- * LC zavisnost postoji ako postoje dva iterativna vektora I i J , $I < J$, takva da važi da je $f(I) = g(J)$
- * problem se svodi na rešavanje sistema celobrojnih jednačina (Diofantove jednačine)
- * Problem je np kompleksan
 - sprovode se jednostavniji testovi koji samo daju dovoljne uslove (ali ne i potrebne) za postojanje zavisnosti

GCD test za jednostrukе petlje

- * Prepostavimo da se u petlji vrši upis u element polja sa indeksom $a*i+b$, a da se pristupa elementu sa indeksom $c*i+d$, gde je i indeks petlje koji se kreće u granicama od m do n .
- * Loop-carry zavisnost postoji ako važi sledeće:
 - Postoje dva iterativna indeksa j i k ($j < k$), oba unutar granica petlje, tako da se u petlji pamti rezultat u element sa indeksom $a*j+b$, a zatim se pribavlja (čita) isti element kada je indeksiran sa $c*k+d$, tj. kada je $a*j+b = c*k+d$.
- * Jednostavan i dovoljan test koji se koristi za detekciju zavisnosti je NZD (najveći zajednički delilac) test.
(GCD)

NZD test

- * Ako loop-carry zavisnost postoji tada $\text{NZD}(c, a)$ mora deliti $(d - b)$ bez ostatka (baš $(d - b)$ jer je $j < k$).
- * NZD test je dovoljan da garantuje da nema nikakve zavisnosti.
- * Može se desiti da NZD test da odgovor da zavisnost postoji a da nema nikakve zavisnosti.
 - Ovo može da nastupi zato što NZD test ne uzima u obzir granice petlje.

GCD Test

- Može se primeniti i kada je indeks polja linearna funkcija indeksnih promenljivih
 - Npr.

```
do i = li,hi  
    do j = lj,hj  
        A(a1*i + a2*j + a0) = ... A(b1*i + b2*j + b0) ...  
    enddo  
enddo
```

- Ako loop-carry zavisnost postoji tada

- $a_1*i_1 - b_1*i_2 + a_2*j_1 - b_2*j_2 = b_0 - a_0$
- Celobrojna rešenja postoje ako
- $\gcd(a_1, a_2, b_1, b_2)$ deli $b_0 - a_0$

Primer 1

- * Korišćenjem NZD testa utvrditi da li postoji loop-cary zavisnost u sledećoj petlji:

for $i=1$ to 100

$$x(2*i+3)=x(2*i)+k$$

endfor

- * Rešenje $a = 2, b = 3, c = 2, d = 0$

$$\text{NZD}(c, a) = \text{NZD}(2, 2) = 2$$

$$d - b = 0 - 3 = -3$$

- Pošto 2 ne deli -3 bez ostatka to nema loop-cary zavisnosti.
- Ako razvijemo petlju videćemo da zavisnosti zaista nema:

$$i=1 \quad x(5)=x(2)+k$$

$$i=2 \quad x(7)=x(4)+k$$

$$i=3 \quad x(9)=x(6)+k$$

Primer2

- * Korišćenjem NZD testa utvrditi da li postoji loop-cary zavisnost u sledećoj petlji:

```
for i=2, 100, 2  
    x(50*i+1)=x(i-1)+k  
endfor
```

* REŠENJE:

- Da bi NZD test mogao da se primeni prvo treba normalizovati petlju, pošto se u primeru indeks petlje uvećava sa korakom 2.
- Normalizovana petlja ima sledeći izgled

```
for i=1, 50  
    x(100*i+1)=x(2*i-1)+k  
endfor
```

$$a = 100, \ b = 1, \ c = 2, \ d = -1$$

$$NZD(c, a) = NZD(2, 100) = 2$$

$$d - b = -1 - 1 = -2$$

NZD test prolazi, mada nema loop carry zavisnosti

Primer2 – nast.

* Zašto je NZD dao odgovor da loop carry zavisnost postoji?

- zato što test ne uzima u obzir granice petlje!

$$i=1 \quad x(101)=x(1)+k$$

.

.

.

$$i=50 \quad x(5001)=x(99)+k$$

$$i=51 \quad x(5101)=x(101)+k$$

zavisnost postoji
izmedju 1. i 51.
iteracije (ali su
granice 1,50 !)

Primer

Example

Code

```
do i = li, hi
    do j = lj, hj
        A(4*i + 2*j + 1) = ... A(6*i + 2*j + 4) ...
    enddo
enddo
```

$$\gcd(4, -6, 2, -2) = 2$$

Does 2 divide 4-1?

Ograničenja NZD testa

* Nije uvek moguće NZD testom utvrditi postojanje zavisnosti:

- S obzirom da se analiza zavisnosti izvodi u fazi kompilacije, vrednosti a , b , c i d mogu biti nepoznate, što znači da se ovaj test može primeniti samo ako su a , b , c i d konstante.
- Ako postoji više ugnježdjenih petlji i ako pristup elementu zavisi od više iterativnih indeksa NZD test se takođe ne može primeniti.
 - U tom slučaju se analiza zavisnosti vrši preko vektora zavisnosti.

Eliminisanje loop carry zavisnosti

- Mnoge loop-carry zavisnosti mogu biti eliminisane preuredjenjem naredbi unutar petlje i razbijanjem petlje.
- **PRIMER.** U sedećoj petlji postoji loop-carry zavisnost izmedju naredbi S1 i S2 kada je u pitanju vektor y :

```
for i=1, N  
S1    x(i)= y(i-1) * z(i)  
S2    y(i)= 2*y(i)  
      endfor
```

- Preuredjenjem ove petlje i razbijanjem na dve mogu se eliminisati loop-carry zavisnosti:

```
x(1)=y(0) *z(1)  
for i=1, N  
y(i)= 2*y(i)  
endfor  
for i=2,N  
x(i) = y(i-1)*x(i)  
endfor
```

- Ovo je moguće jer izračunavanje $y(i)$ ne zavisi od $x(i)$, tj. ne postoji kružna zavisnost izmedju naredi S1 i S2 u polaznoj petlji.

Eliminacija zavisnosti

- U petlji mogu postojati i zavisnosti tipa WAR koje se još zovu i antizavisnosti, WAW koje se zovu izlazne zavisnosti.
- Međutim, ove zavisnosti kod vektorski računara nisu parvi hazardi i mogu se eliminisati preimenovanjem registara u fazi kompilacije.
- Ovi hazardi se zovu još i *konflikti imenovanja*

```
do 10 i = 1, 100
  > 1      y(i) = x(i)/S
  > 2      x(i) = x(i) + S
  > 3      z(i) = y(i) + S
  > 4      y(i) = S - y(i)
10      continue
```

```
graph TD
    1((> 1)) --> 4((> 4))
    2((> 2)) --> 3((> 3))
    2((> 2)) --> 4((> 4))
```

• Prave zavisnosti (RAW) postoje izmedju naredbi 1 i 3, i 1 i 4, zbog y(i).

- Ove zavisnosti nisu loop-carry i neće sprečiti vektorizaciju.
- Ove zavisnosti usloviće da naredbe 3 i 4 čekaju dok se naredba 1 ne završi, mada ne koriste iste funkcionalne jedinice;

• WAR (antizavisnosti) postoje izmedju 1 i 2 zbog x(i), i 3 i 4 zbog y(i);

• WAW (izlazna zavisnost) postoji izmedju 1 i 4 zbog y(i).

- Sledećom verzijom petlje mogu se eliminisati ove pseudozavisnosti:

•	do 10 i = 1, 100
• 1	T(i) = x(i)/S
• 2	X1(i) = x(i) + S
• 3	Z(i) = T(i) + S
• 4	Y(i) = S - T(i)
10	continue

Analiza zavisnosti i vektorizacija ugnježdjenih petlji

* Kod vektorizacije gnezda petlji vrši se vektorizacija najdublje ugnježđene petlje.

- Zbog činjenice da sada pristup elementu polja zavisi od više iterativnih indeksa analiza zavisnosti po podacima se sada ostvaruje preko vektora zavisnosti.

vektorizacija ugnježdjenih petlji (nast.)

➤ Razmotrimo jedno ovakvo gnezdo petlji

```
for I1 = l1, u1
  for I2 = l2, u2
    :
    for In = ln, un
      S1(I)
      S2(I)
      :
      Sk(I)
»      endfor
```

* gde su I_j, u_j granice petlje:

- to mogu biti celobrojni izrazi u kojima figurišu indeksi I_1, I_2, \dots, I_{j-1} .
- I je uredjena n-torka indeksa (I_1, \dots, I_n) .
- S_1, \dots, S_k su naredbe dodeljivanja u kojima se pojavljaju 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, \dots, 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}
```

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

matrica
zavisnosti

* REŠENJE:

- Uočimo prvo sve parove generisanih—korišćenih promenljivih:

$$A(i, j, k) \text{ i } A(i-1, j, k+1)$$

$$B(i, j, k+1) \text{ i } B(i, j, k),$$

$$B(i, j, k+1) \text{ i } B(i, j-1, k-1)$$

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 $\neq 0$ nosi zavisnost!

vektorizacija ugnježdjenih 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:

$$\text{pravac_zavisnosti} = \begin{cases} <, & \text{if } d_i > 0 \\ =, & \text{if } d_i = 0 \\ >, & \text{if } d_i < 0 \end{cases}$$

- * Za prethodne vektore zavisnosti odgovarajući pravci zavisnosti su

- $d_1 = [1, 0, -1]^T \Rightarrow [<, =, >]^T$
- $d_2 = [0, 0, 1]^T \Rightarrow [=, =, <]^T$
- $d_3 = [0, 1, 2]^T \Rightarrow [=, <, <]^T$

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

Pravilo

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

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

- vektorizacija po indeksnoj promenljivoj k nije moguća jer za drugi vektor zavisnosti postoji loop-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 $k=1, 2, \dots, 20$ imamo

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

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

loop-carry zavisnost
- $i=1, j=1, k=3 \quad A(1, 1, 3) = A(0, 1, 4) + B(1, 1, 3)$
- $B(1, 1, 4) = B(1, 0, 2) * 3$

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 nenulti 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

$$I_1 = l_1, u_1$$

$$I_2 = l_2, u_2$$

.

.

$$I_m = l_m, u_m$$

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

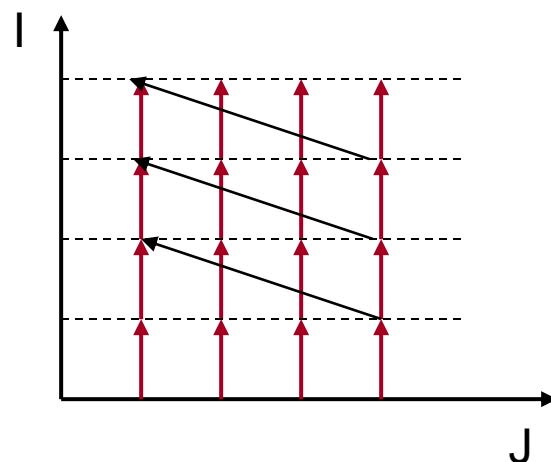
$$\begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} I \\ J \end{bmatrix} = \begin{bmatrix} J \\ I \end{bmatrix}$$

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

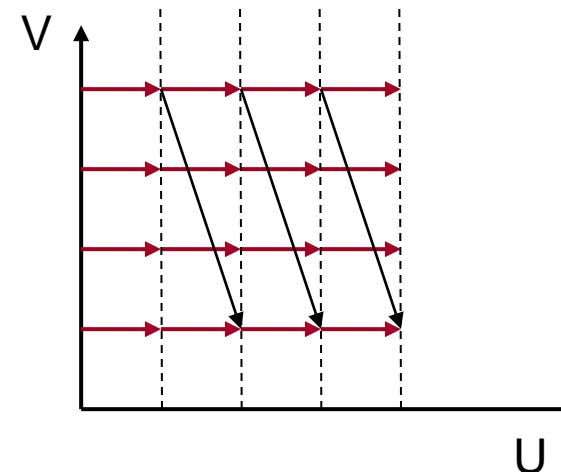
```
for 10 U = 1, n  
    for 10 V = 1, n  
        A(U) = A(U) + C(V, U)  
    endfor{u,v}
```

Transformacija permutacije

- Transformacijom je promenjen redosled izračunavanja elemenata vektora A.



Pre transformacije



Posle transformacije

Transformacija obrtanje

- * Omogućava promenu redosleda izračunavanja po određenoj 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 применимо transformaciju obrtanja po indeksnoj promenljivoj j .
- Matrica transformacije je oblika

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

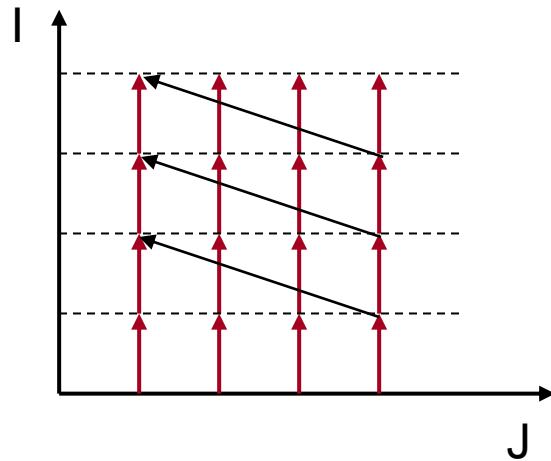
Obrtanje – primer (nast.)

- Novi indeksni skup dobijamo na sledeći način

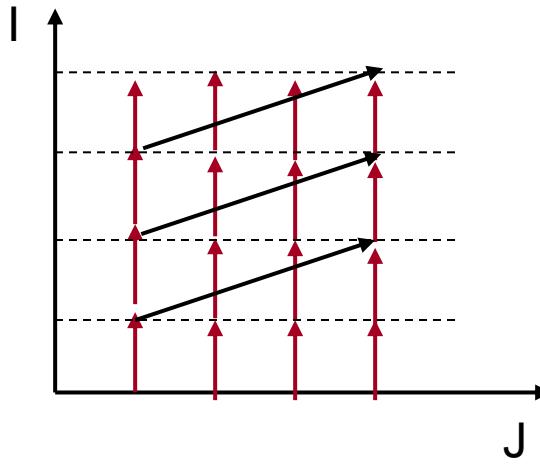
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ -j \end{bmatrix}$$

- 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 (skewing) jednog iterativnog indeksa u odnosu na drugi za faktor **f**.
- Prepostavimo da imamo ovakvu iteraciju indeksnih promenljivih

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

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

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

Kriviljenje

* Kriviljenje petlje i_k u odnosu na i_j za faktor f

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

Kriviljenje – 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 kriviljenje 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
```

Kriviljenje – primer (nast.)

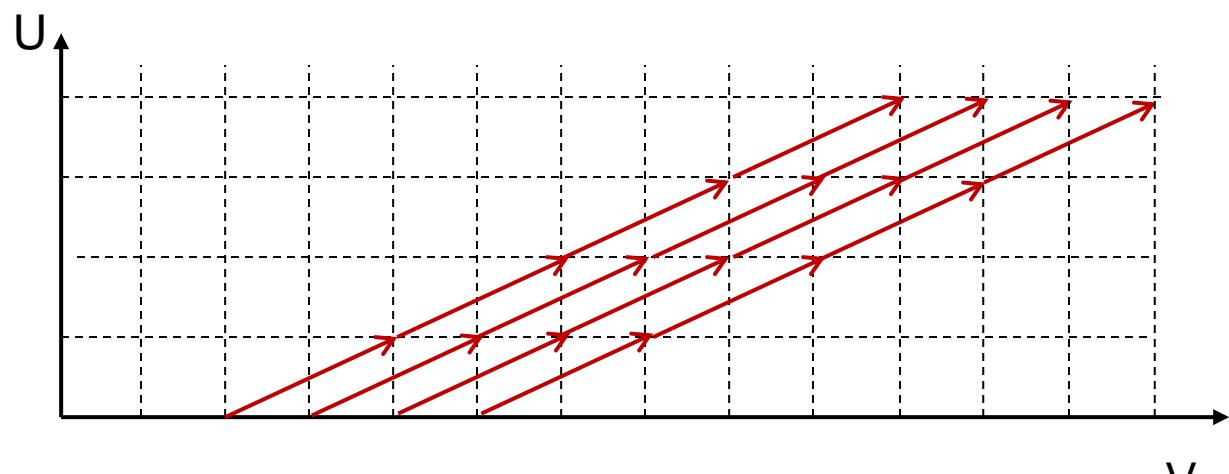
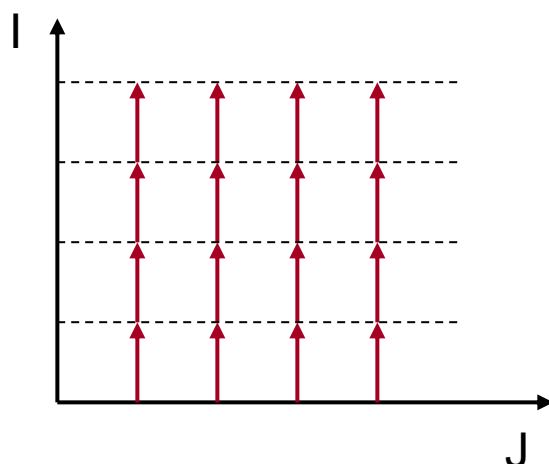
* Transformisana petlja ima sledeći izgled

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}



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}
```

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

matrica
zavisnosti

* REŠENJE:

- Uočimo prvo sve parove generisanih—korišćenih promenljivih:

$$A(i, j, k) \text{ i } A(i-1, j, k+1)$$

$$B(i, j, k+1) \text{ i } B(i, j, k),$$

$$B(i, j, k+1) \text{ i } B(i, j-1, k-1)$$

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 d_2 nije moguće izvršiti vektorizaciju.

- permutacijom petlji j i k dobićemo kod koji je moguće vektorizovati

Primer-1

* Matrica zavisnosti pre permutacije

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

* Matrica zavisnosti nakon permutacije

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

* 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 takođe 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:

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

- 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

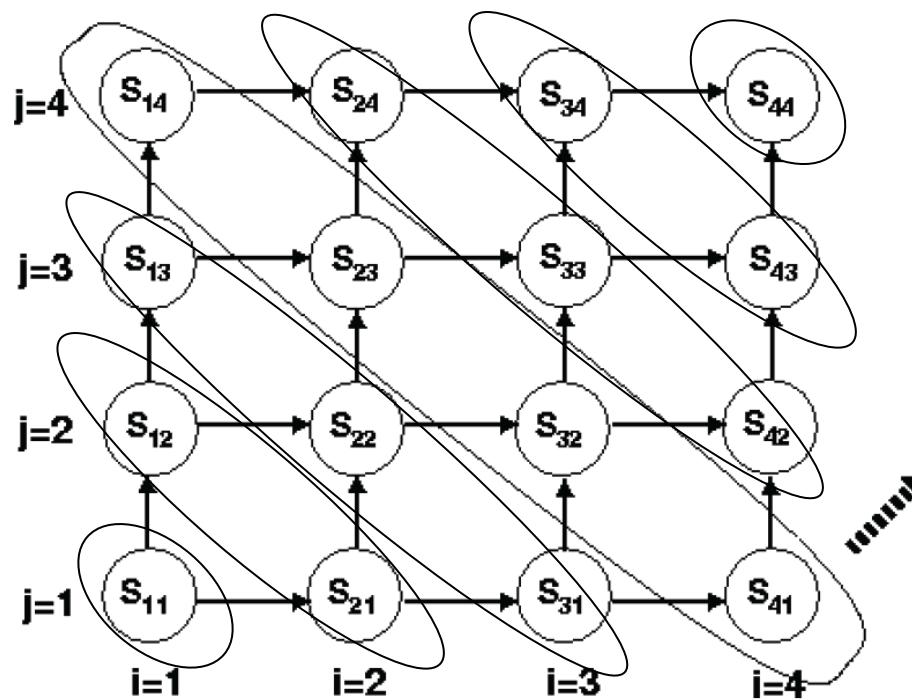
v=3 izračunava se A(3,1)=A(2,2)*2

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?

Tehnike za poboljšanje performansi

- * Konvoj
- * Ulančavanje (Chaining)
- * Uslovno izvršenje (Conditional execution)

Konvoj

* Konvoj

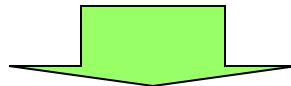
- Predstavlja skup vektorskih instrukcija čije izvršenje može otpočeti u istom clk periodu
- Za instrukcije koje se nalaze u konvoju ne smeju da postoje strukturni hazardi ili hazardi po podacima

* Zvono (Chime)

- Aproksimativna mera vremena izvršenja niza vektorskih instrukcija

Primer konvoja

LV	V1, Rx	; load vektor X
MULVS	V2, F0, V1	; množenje vektora skalarom
LV	V3, Ry	; load vektor Y
ADDV	V4, V2, V3	; add
SV	Ry, V4	; store rezultata



1. LV
2. MULSV LV
3. ADDV
4. SV

Chime = 4
 $VL=64$
 $\Rightarrow 4 \times 64 = 256 \text{ clocks}$
(ili 4 cik po rezultatu)

Convoy

Primer konvoja

Unit	Start-up overhead
Load and store unit	12 cycles
Multiply unit	7 cycles
Add unit	6 cycles

Latentnosti funkcionalnih jedinica

Konvoj: grupa nezavisnih vektorskih instrukcija koje se mogu izvršavati jednovremeno, konvoji se ne preklapaju, dužina vektora n

Convoy	Start	1st result	last result	
1. LV	0	12	$11+n (=12+n-1)$	
2. MULVS.D	$12+n$	$12+n+7$	$18+2n$	Mult. start-up
LV	$12+n+1$	$12+n+13$	$24+2n$	Load start-up
3. ADDV.D	$25+2n$	$25+2n+6$	$30+3n$	Wait convoy 2
4. SV	$31+3n$	$31+3n+12$	$42+4n$	Wait convoy 3

$$n=64, (42+4*64)=4,65 \text{ clk po rezultatu}$$

Ulančavanje

* Ulančavanje

- * Omogućava da vektorska operacija otpočne sa izvršenjem čim jedan element izvornog vektorskog operanda postane dostupan (umesto da se čeka da ceo vektor bude izračunat i upisan u vektorski registar)
- Dozvoljava da operacije budu smeštene u isti konvoj i na taj način redukuje vreme izvršenja (chime).

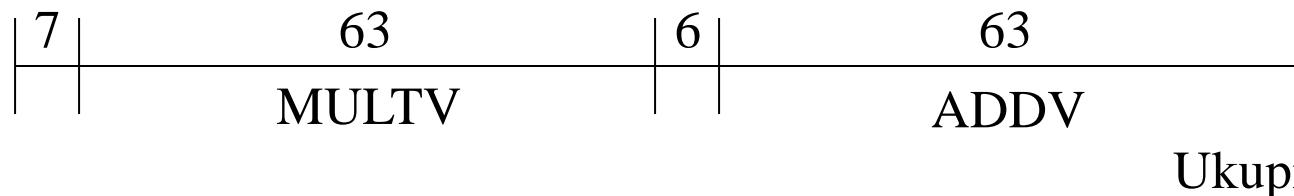
Primer ulančavanja

$$Y = a^* X + Y$$

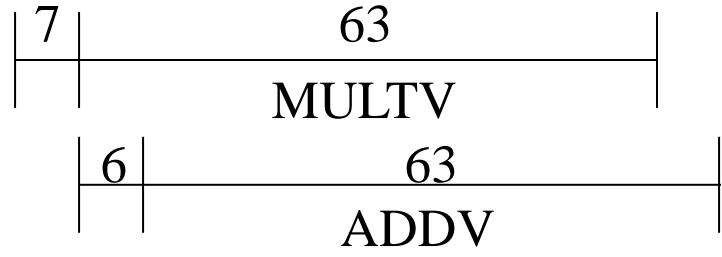
Funkcionalna jedinica	latentnost
Multiply unit	7 cycles
Add unit	6 cycles

MULTV V1, V1, F0
ADDV V2, V1, V2

**bez
ulančavanja**

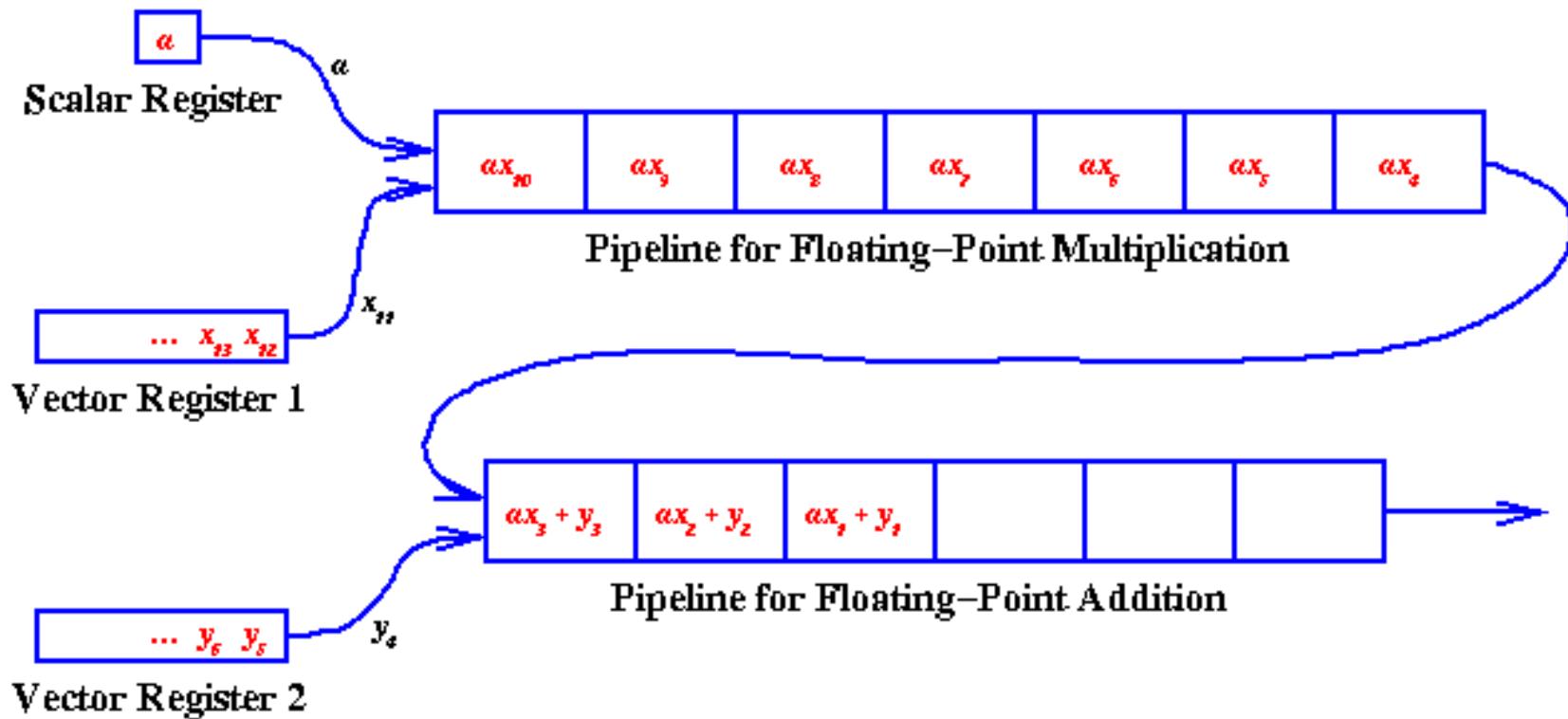


**Sa
ulančavanjem**



Ukupno = 76

Primer ulančavanja

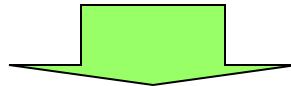


Uslovno izvršenje

- * Petlje koje sadrže naredbe uslovnog grananja se ne mogu vektorizovati direktno.
- * Registr vektora maske
 - Bilo koja vektorska instrukcija se izvrševa samo za one elemente vektora za koje je odgovarajući bit u vektoru maske postavljen na 1.
 - Ne skraćuje se vreme izvršenja, već samo omogućava vektorizaciju, ali je vektorsko izvršenje brže od skalarnog izvršenja.

Primer uslovnog izvršenja

```
for i = 1, 64
    if (A(i) .ne. 0) then
        A(i) = A(i) - B(i)
    endif
endfor{i}
```



LV	V1, Ra	; load vektor A u V1
LV	V2, Rb	; load vektor B
LD	F0, #0	; load FP 0 u F0
SNESV	F0, V1	; postavi VM na 1 ako je V1(i) ≠ F0
SUBV	V1, V1, V2	; oduzimanje pod maskom VM
CVM		; postavi VM na sve 1 (brisanje VM)
SV	Ra, V1	; zapamti rezultat u A

Strip Mining

- ❖ **Strip mining** – razbijanje jedne petlje na dve ugnježđene petlje zbog ograničenja u veličini vektorskih registara
 - ✓ Kada je dužina vektora veće od dužine vektorskog регистра (ili je gornja granica petlje promenljiva), vrši se segmentiranje vektora na segmente fiksne veličine (MVL, maximum vector length).

Primer Strip Mining

```
for i = 1, n  
    Y(i) = a * X(i) + Y(i)
```



```
low = 1  
VL = (n mod MVL)  
for j = 0, (n/MVL)  
    for i = low, low+VL-1  
        Y(i) = a*X(i) + Y(i)  
    endfor{i}  
    low = low + VL  
    VL = MVL  
enfor{j}
```

; nalazi segment $VL < MVL$
; spoljnja petlja
; prvo izvršenje je za $VL < MVL$
; glavno izračunavanje

; def. nove donje granice
; resetuj dužinu na maximalnu
; vrednost