Vektorski računari

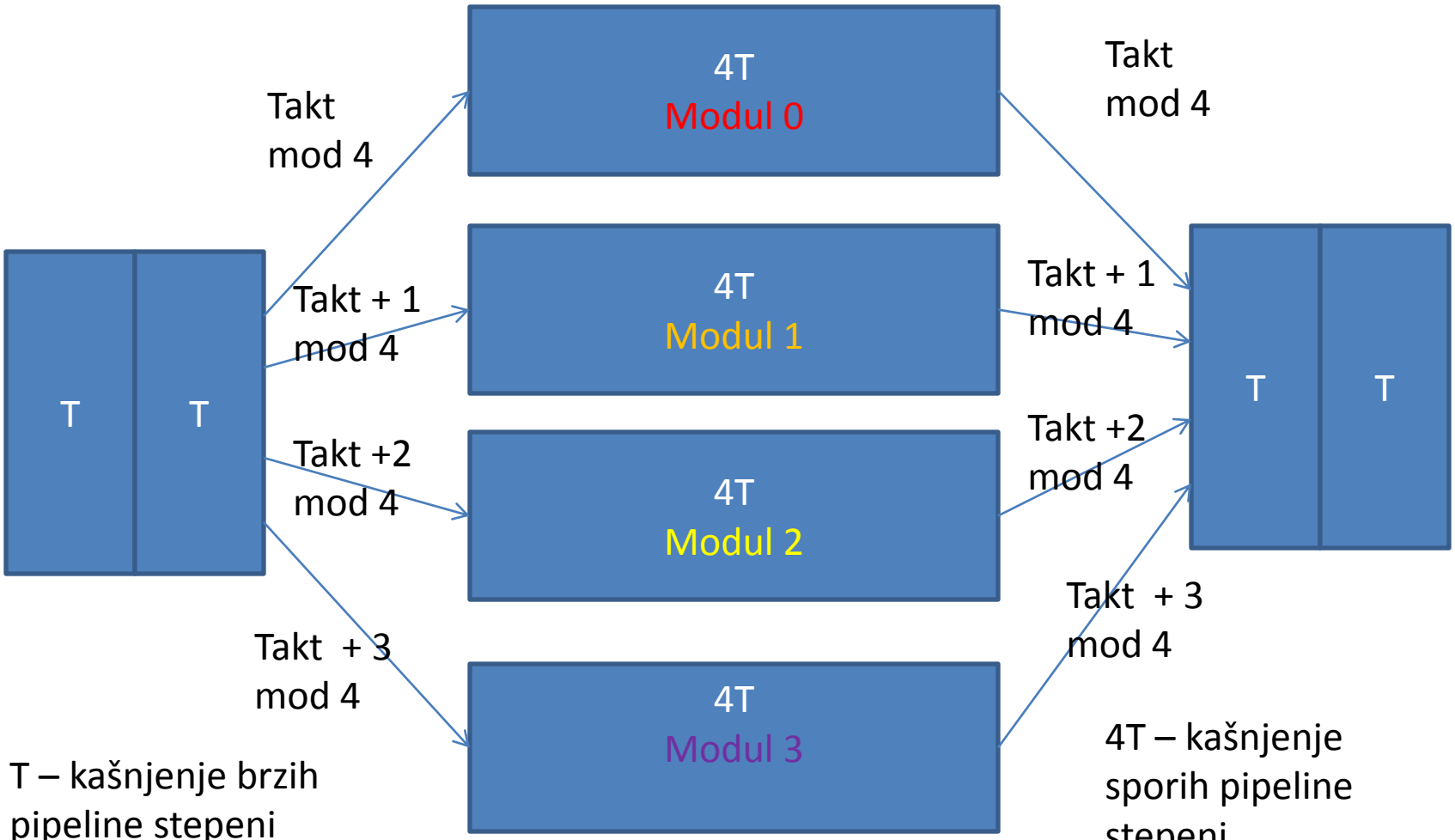
Zoran Jovanovic

Korišćeni neki slajdovi sa vodećih univerziteta u SAD

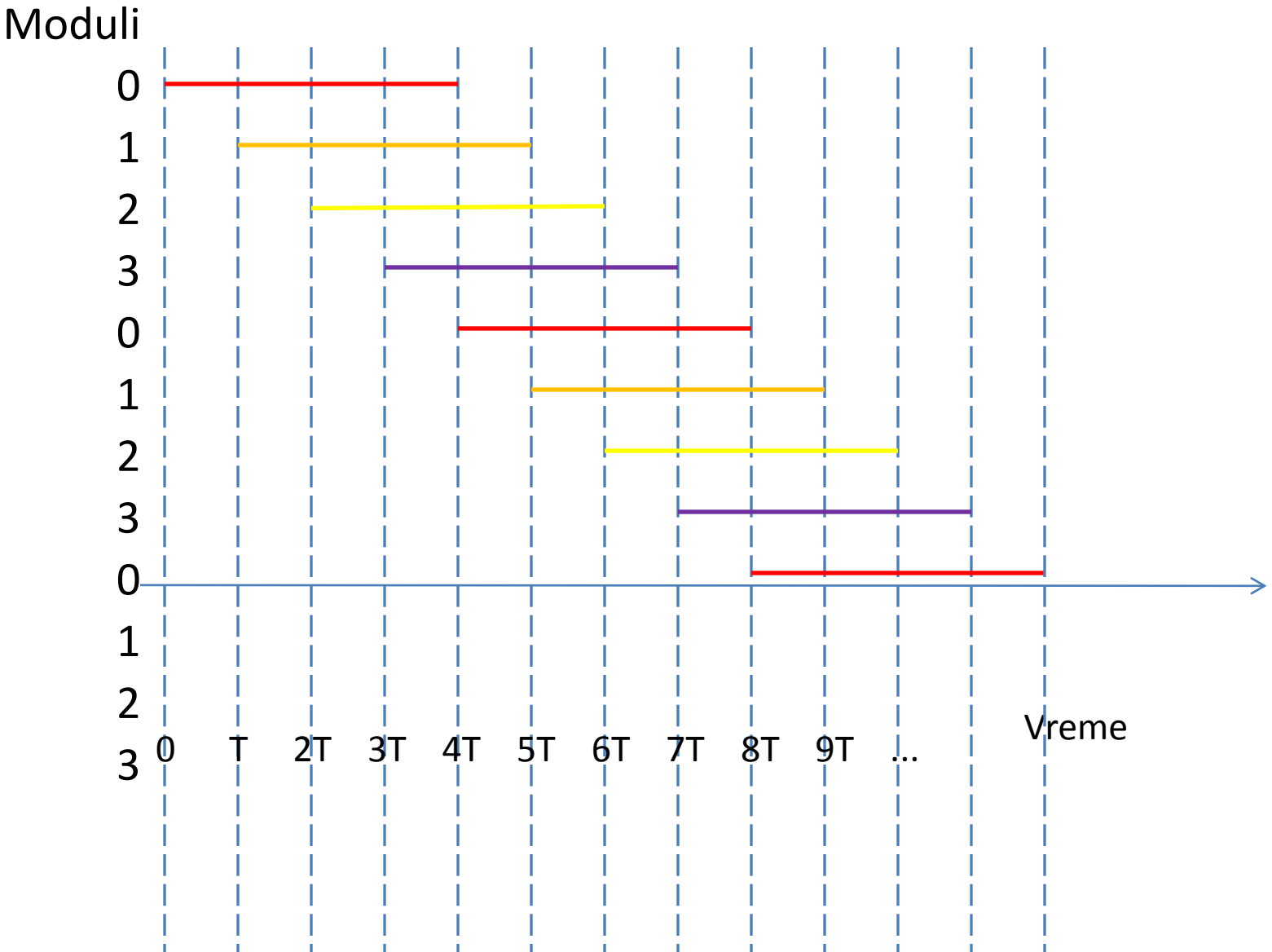
Paralelizam je u DoAll petljama?

- Odvojimo deo mašine za skalarni deo koda sa neregularnim paralelizmom i uradimo neku paralelizaciju (skalarni procesor)
- Drugi deo mašine odvojimo samo za programske petlje, realno za DoAll i napravimo da postoji hardverska podrška za nizove (vektorski procesor)
- Svaki deo radi ono za šta je bolji, a kompajler ili čak programer odvaja kôd za skalarni ili vektorski procesor
- Kako napraviti vektorski procesor?

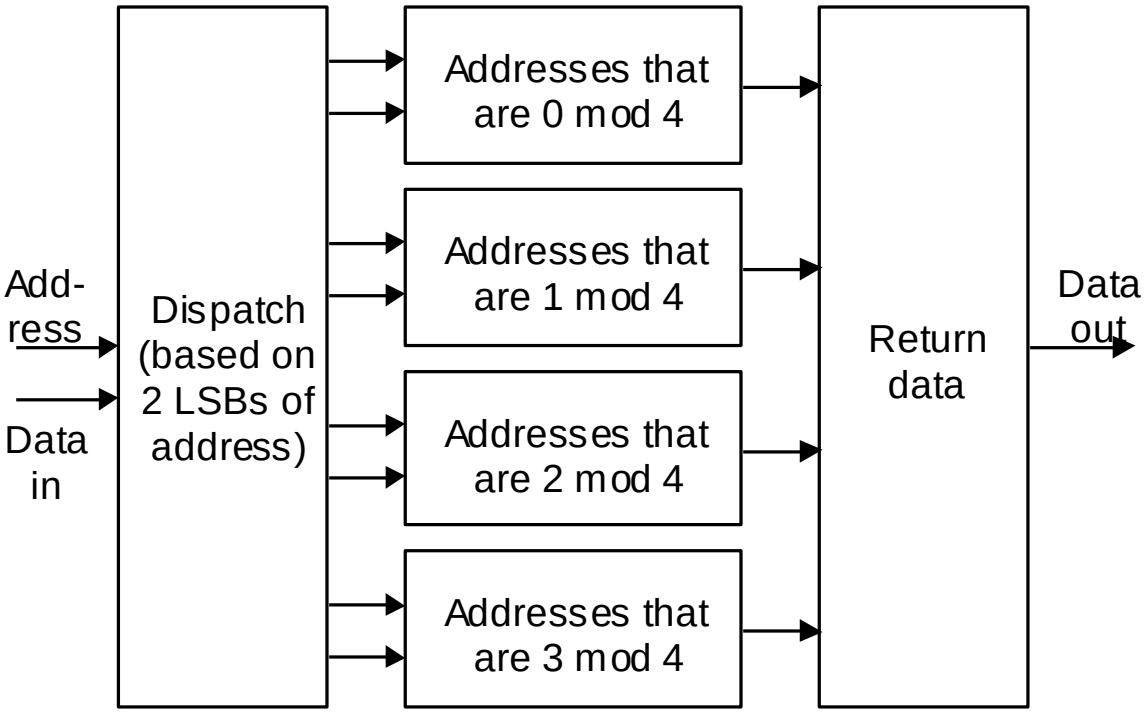
Protočni stepeni sa velikim kašnjenjem kombinacione logike u bržem pipeline-u

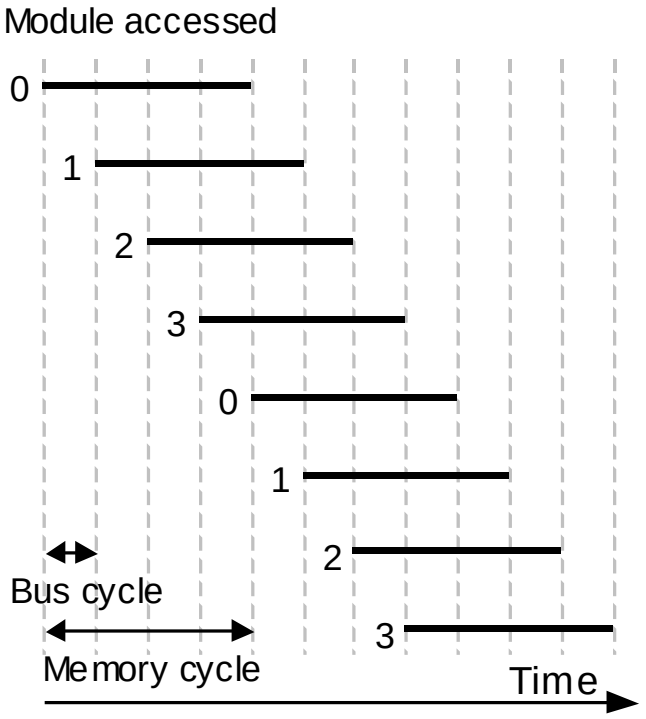


Ciklusi

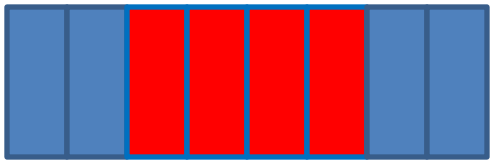


Memory Interleaving – memorija je spora!!!





Ako se sekvencijalno ide po adresama u memorijskim operacijama, iluzija je sledeća:

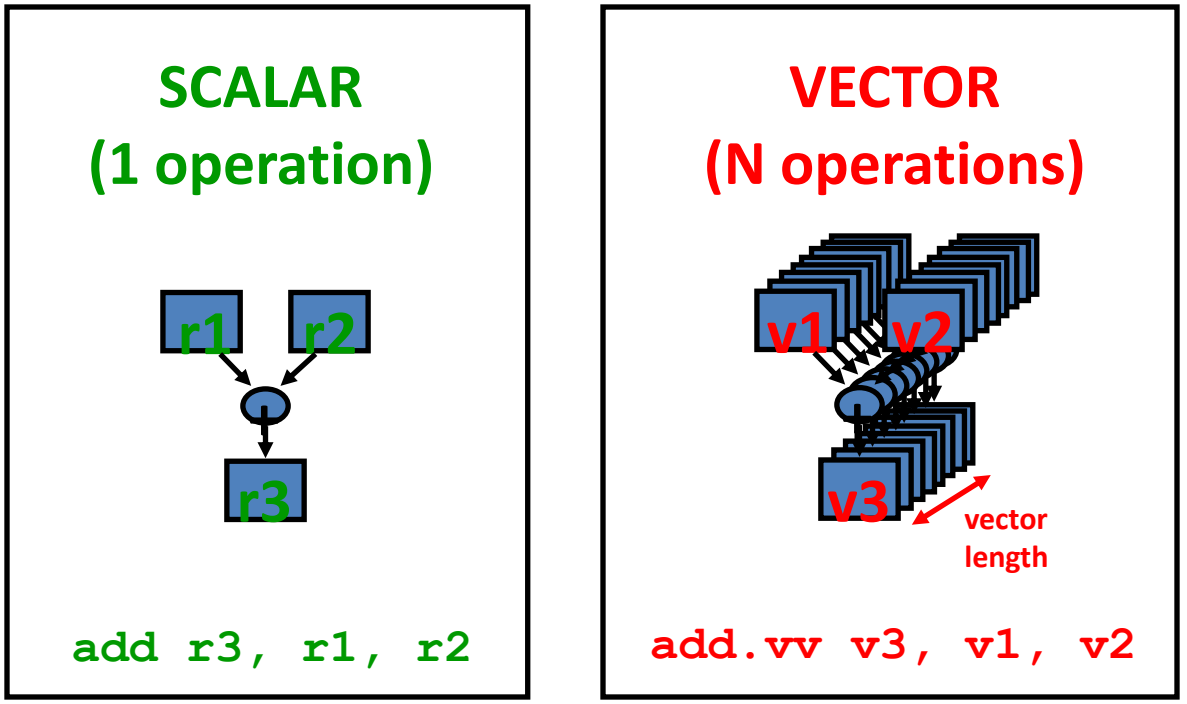


"Memorijski pipeline"

4* sporija memorija radi efektivno punom brzinom u pipeline-u ako čitamo ili upisujemo u blokove podataka!!!
CRAY 16* sporija memorija (mod 16)!!

Model za petlje koje obrađuju nizove podataka – naročito DoAll

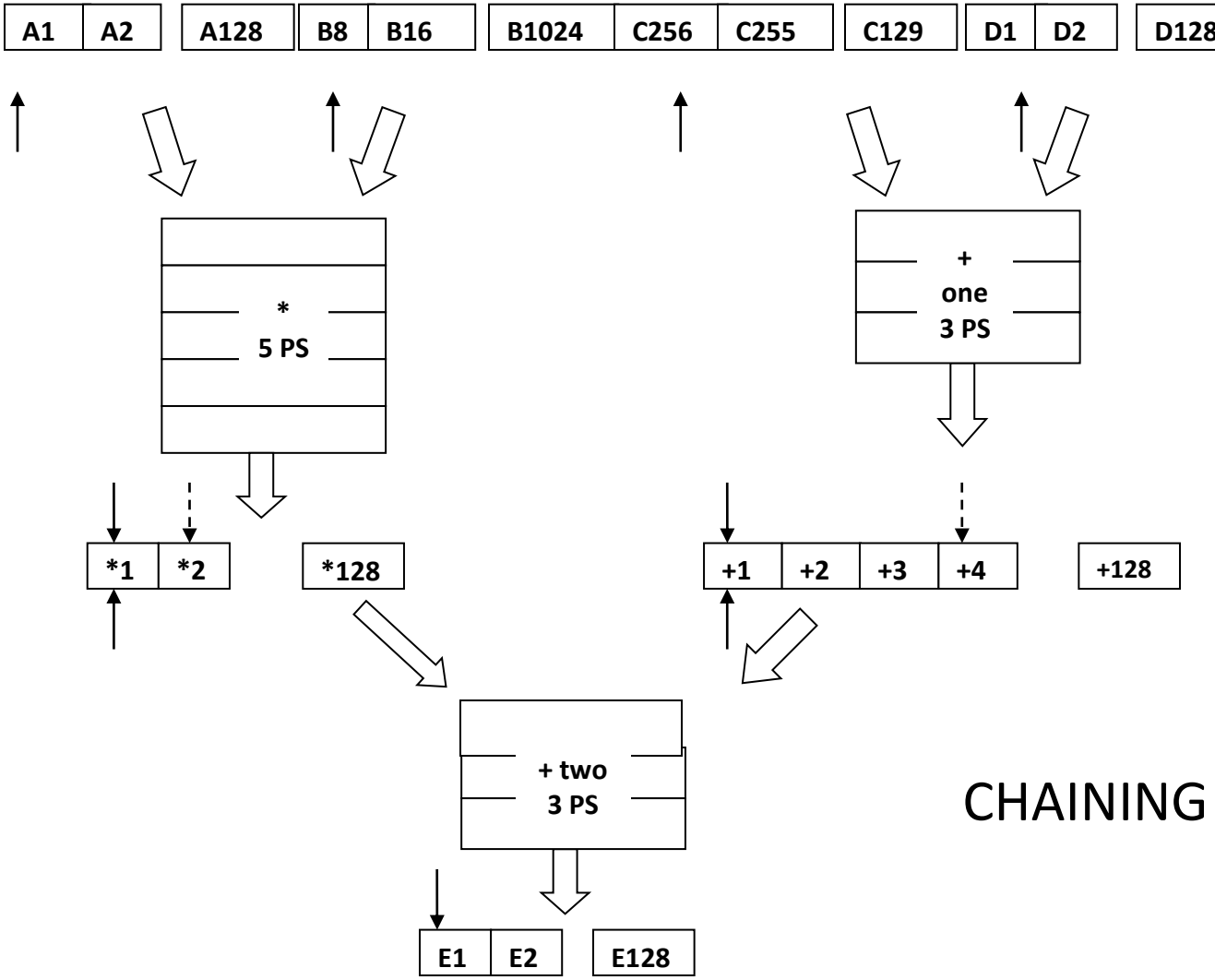
- Iskoristiti Memory interleaving da se napune vektorski registri (sadrže cele nizove za sve ili deo operacija data ready na vrhu grafa)
- Međurezultati se takođe čuvaju u vektorskim registrima



Komponente CRAY Vektorskih Procesora

- *Vektorski Registri*: Fiksne veličine za jedan vektor (niz)
  - ima najmanje 2 read i 1 write port
  - tipično je 8-32 vektorska registra, svaki sa 64-128 64-bit elemenata
- *Vektorske Funkcionalne Jedinice (FJ)*: pipeline koji započinje novu operaciju svakog ciklusa
  - tipično 4 to 8 FJ: npr. 2x FP add, 2x FP mult, FP reciprocal (1/X), integer add, logical, shift;
- *Vector Load-Store jedinice (LSUs)*: memorijski pipeline interleaved memorije
- *Skalarni registri*: po jedan element za FP skalar ili adresu
- Cross-bar da poveže FJ , LSU, registre

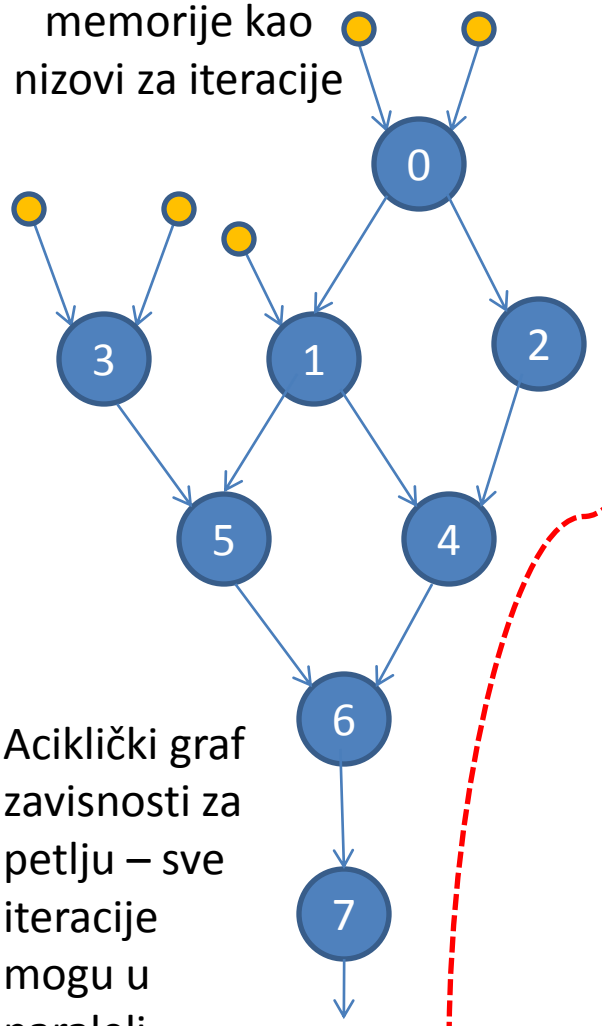
Do I = 1,128 E(I) = A(I)*B(8*I)+C(257-I)+D(I);



CHAINING

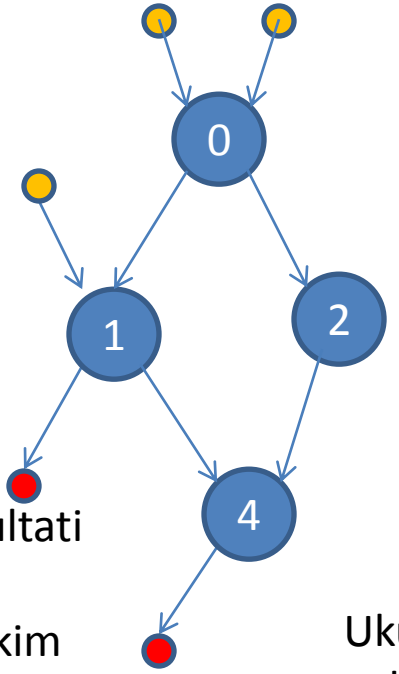
Chaining, graf DoAll i vektorski registri

Argumenti iz memorije kao nizovi za iteracije



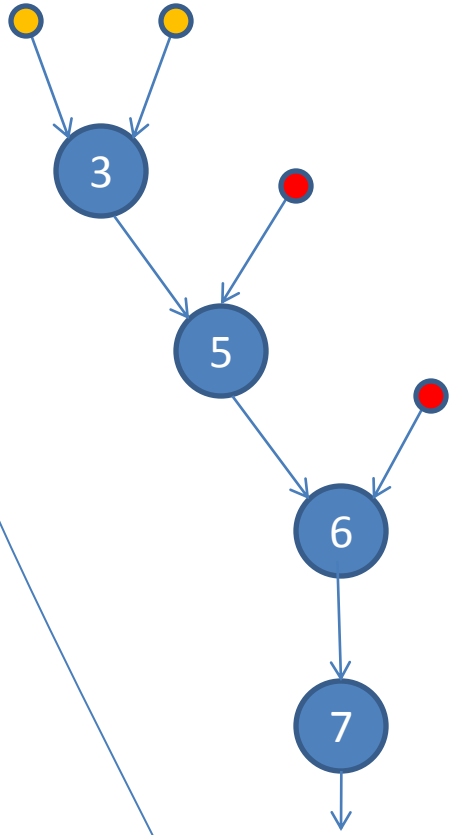
Aciklički graf zavisnosti za petlju – sve iteracije mogu u paraleli

Ideja: prvo uraditi za sve iteracije deo grafa (~4 operacije se započinje po ciklusu)



Ovi rezultati ostaju u vektorskim registrima

Zatim uraditi sledeći deo (~4 operacije po ciklusu) ...



Ukupno 7 vektorskih registara

Vektorska jedinica i ulančavanje instrukcija chaining

- Iz acikličkog grafa otkidamo čitave podgrafove slobodne na vrhu i povezujemo ih u složene pipeline-e prema grafu zavisnosti po podacima DoAll petlje.
- Data ready nizove ubacujemo u vektorske registre – **Interleaving. Nema cache za nizove!!!**
- Veličina podgrafa zavisi od broja raspoloživih ALU i broja vektorskih registara
- Veza rezultat-argument ostvaruje se preko istog vektorskog registra
- Vektorski registri mogu da ubace kašnjenja (pointer(i) za read i write pomereni) da se sinhronizuje stizanje argumenata za ulančani pipeline

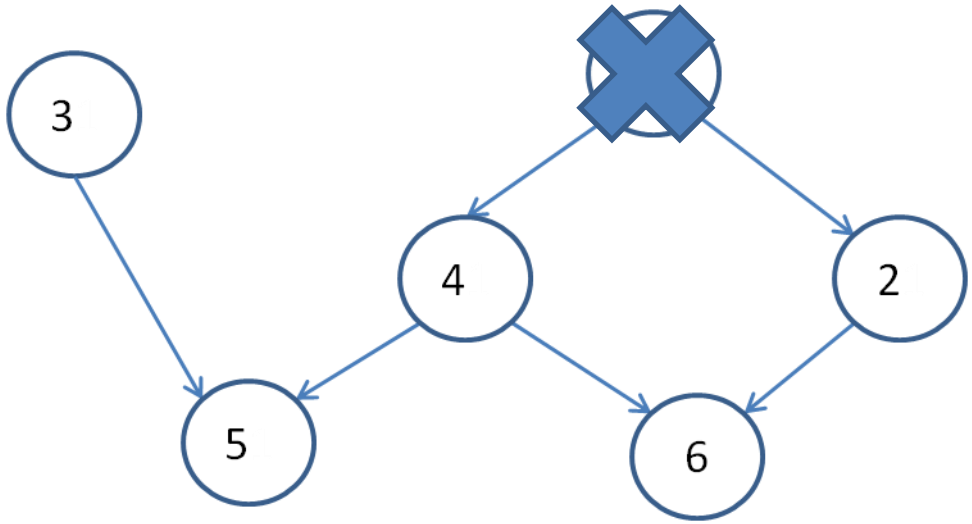
Vektorska jedinica i ulančavanje instrukcija (2)

- Deo argumenata za nove podgrafove koji postaju slobodni na vrhu su već u vektorskim registrima nakon završetka prethodnog podgrafa!!
- Nema adresiranja memorije ili cache-a i povećava se dubina pipeline-a
- Broj lokacija vektorskih registara za elemente nizova je isti
- Ako je broj lokacija svakog vektorskog registra manja od ukupnog broja iteracija, mora se raditi manji broj iteracija = broju lokacija vektorskih registara

SIMD vektorski procesori - pojedinačne instrukcije za veliki broj iteracija

- Operacije slobodne na vrhu se izvršavaju pojedinačno, ali nad većim skupom iteracija istovremeno.
- Izračunati rezultati se vraćaju u vektorske registre i/ili memoriju
- Једноставнија реализација компајлера

Graf DoAll petlje

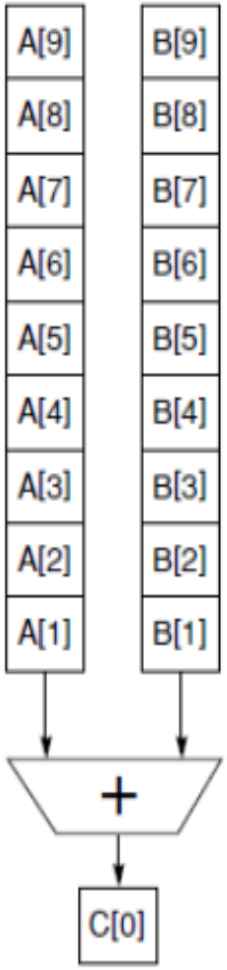


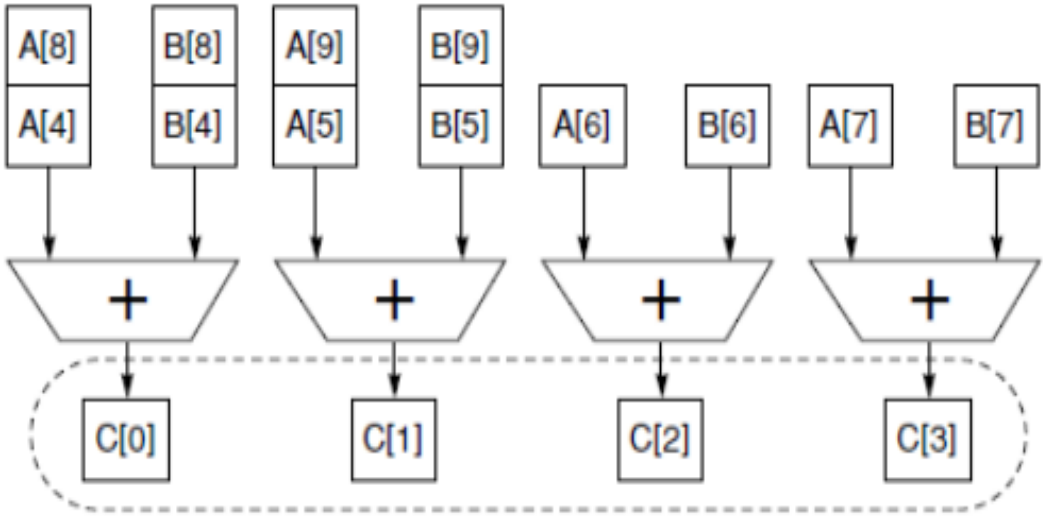
NPR. Instrukcija 1 za veliki broj iteracija

Trake

- Pokazuju odlične performanse za izvršavanje programa sa visokim nivoom paralelizma na nivou podataka
- U svakoj traci se izvršavaju identične operacije nad delom vektora koji predstavljaju argumente operacije i koji su učitani u tu traku
- Jednostavnost realizacije – identične kontrolne sekvence koje upravljaju trakama
- Pogodnost da prevodilac može lako da se prilagođava broju traka bez izmene vektorskih asemblerskih instrukcija

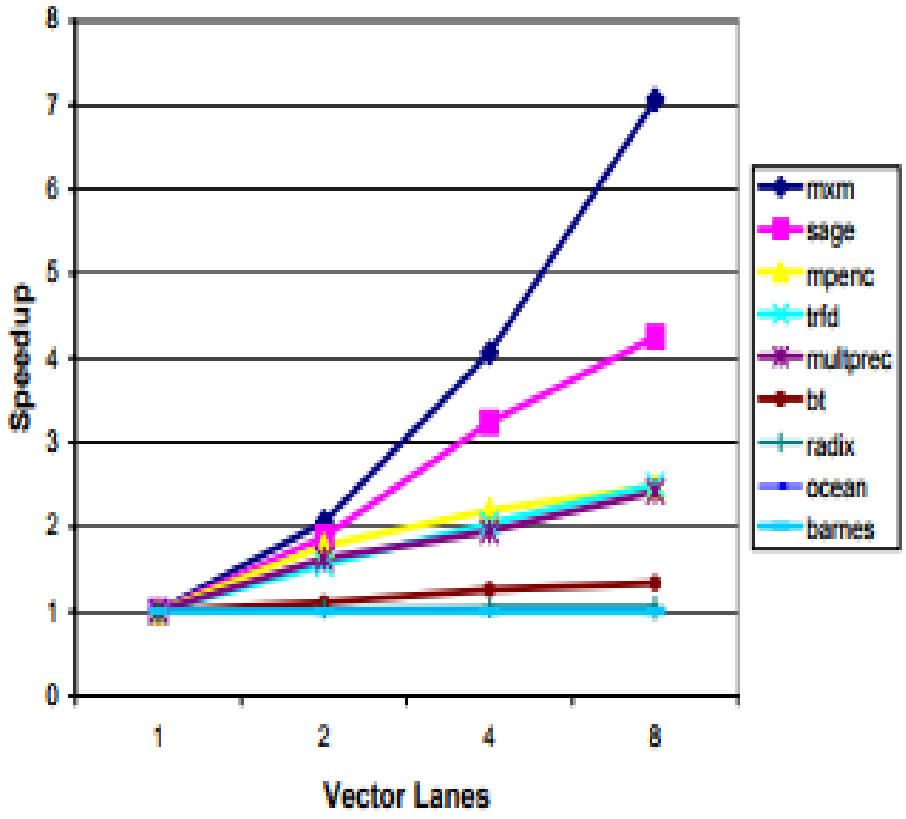
Chaining i trake pojednostavljeno





Trake ubrzanje

- Izvršavanje aplikacije u kojoj vremenski dominira izvršavanje DoAll petlje. Može se značajno ubrzati izračunavanje povećanjem broja traka, ako je dovoljan broj iteracija. Brzina raste samo malo sporije od linearne sa brojem traka uz interfejs od 512 bita.



Predikatsko izvršavanje i SIMD

- Postojanje kontrolnih zavisnosti (grananja) u iteraciji petlje zahteva povećanje kompleksnosti da bi se izvršila vektorizacija
- Na osnovu predikata se određuje da li će operacija biti izvršena i rezultat upisan, za elemente vektora – IF konverzija
- Sastavni deo svake banke su pored vektorskih registara i vektorskih jedinica i predikatski registri u predikatskom registarskom fajlu i predikatske logičke jedinice PLU

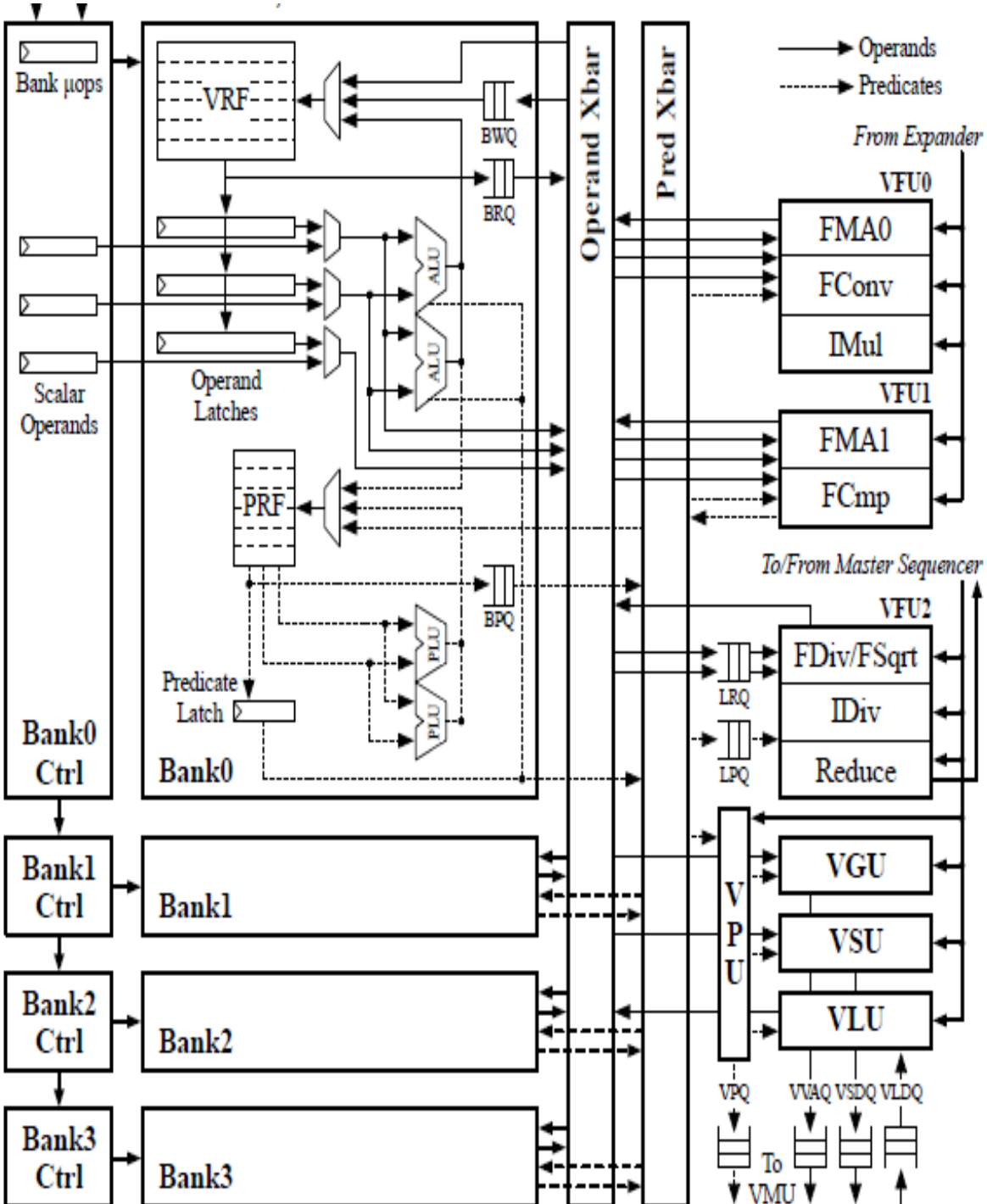
Predikatsko izvršavanje i SIMD (2)

- Ukupan broj bita u predikatskom registru jednak je maksimalnom broju elemenata niza koji se može nalaziti u reči npr. (8)
- Potreban broj bita u predikatskom registru zavisi od broja bita argumenata u odnosu na širinu reči vektorskih registara i određen je konfiguracijom

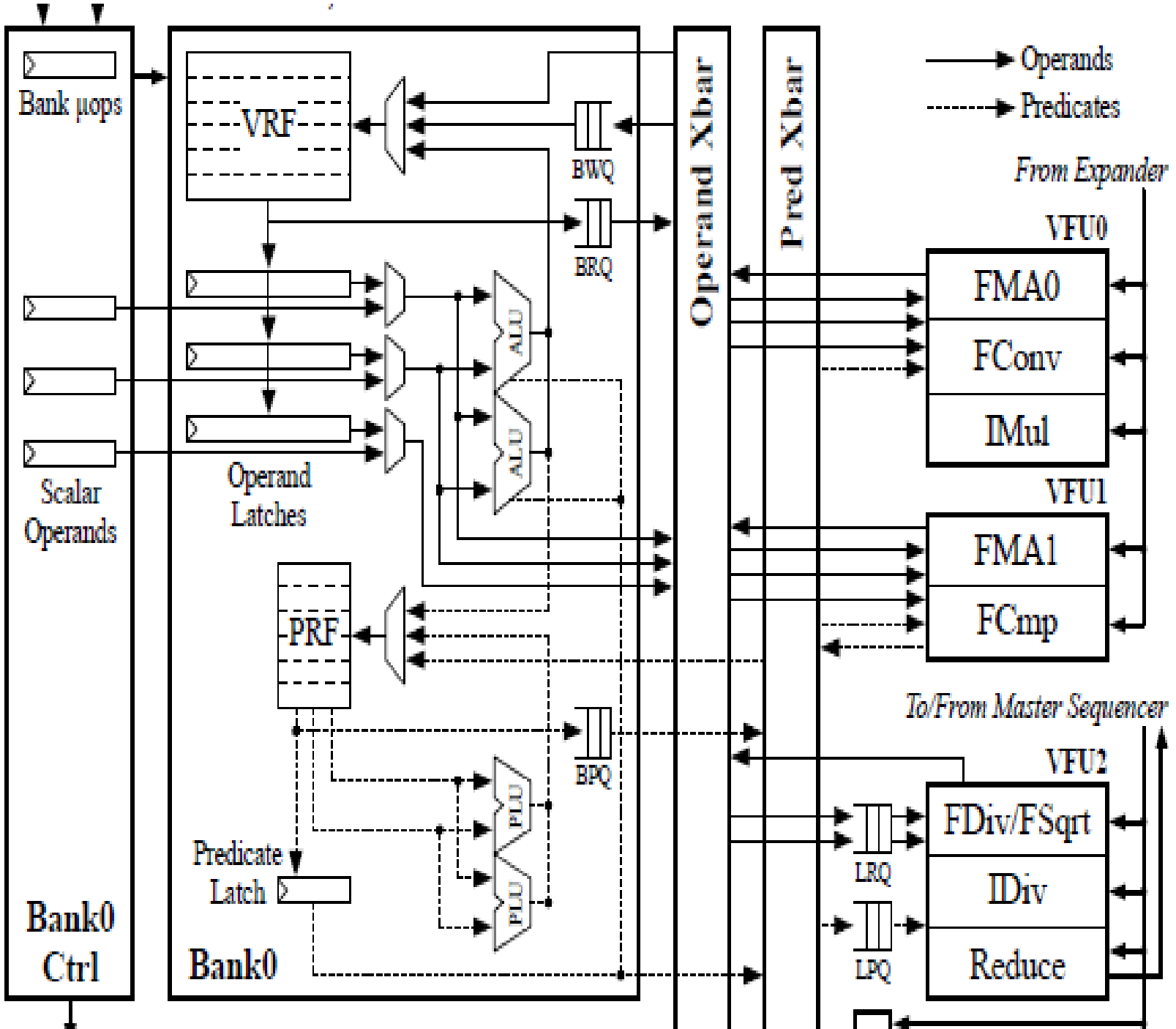
Berkeley 2016 Hwacha

- Akademski razvoj nove generacije vektorskih procesora – simulirala asistent Maja Vukasović za master tezu
- Iz keš memorije se učitava odgovarajući broj elemenata za argumente (operande) instrukcije i prosleđuje se prema utvrđenom rasporedu izabranim trakama
- U trakama se nalaze banke, a u bankama vektorski registri za čuvanje vektorskih argumenata za iteracije dodeljene banci kao i logika za predikatsko izvršavanje
- Izračunavanja, izuzev za FP, se obavljaju u ALU jedinicama u okviru banaka, a samo FP se emituje preko crossbara do FP jedinica

Jedna traka
4 banke



Jedna banka


Bank µops
VRF
BWQ
BRQ
Operand Xbar
Pred Xbar
Operands
Predicates
From Expander
VFU0
FMA0
FConv
IMul
ALU
ALU
Scalar Operands
Operand Latches
VFU1
FMA1
FCmp
PRF
To/From Master Sequencer
VFU2
BPQ
PLU
PLU
Predicate Latch
LRQ
FDiv/FSqrt
IDiv
LPQ
Reduce
Bank0 Ctrl
Bank0

SWAR
(SIMD Within A Register)

Za interpretaciju koliko elemenata niza ima u 128-bitnoj reči vektorskog registra, potrebno je dostaviti konfiguraciju u vreme prevođenja (to treba da radi programski prevodilac)

Preslikavanje elemenata niza se izvodi postavljanjem konfiguracionog vektora

Primer konfiguracionog vektora

Vektorskog registarsko fajla:

{4, 4, 8, 0}

4 * 64 bita, 4 * 32 bita, 8 * 16 bita i 0 * 8 bita

<table>
<tr><td colspan="4">A[1]</td><td colspan="4">A[0]</td></tr>
<tr><td colspan="4">B[1]</td><td colspan="4">B[0]</td></tr>
<tr><td colspan="2">C[3]</td><td colspan="2">C[2]</td><td colspan="2">C[1]</td><td colspan="2">C[0]</td></tr>
<tr><td>D[7]</td><td>D[6]</td><td>D[5]</td><td>D[4]</td><td>D[3]</td><td>D[2]</td><td>D[1]</td><td>D[0]</td></tr>
</table>

SWAR (Polimorfizam)

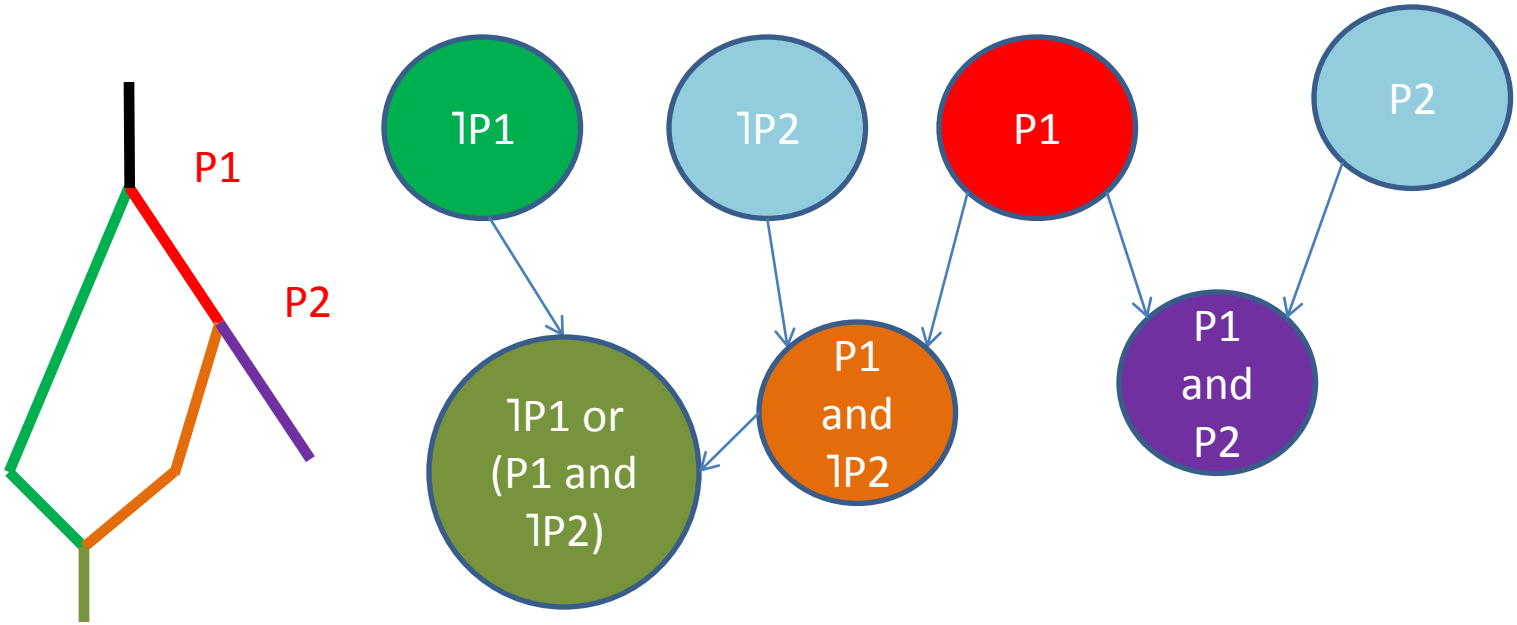
- Pakovanje podataka u vektorske registarske fajlove svih traka događa se implicitno, na osnovu podešene konfiguracije i transparentno je za program!
- Kompajler adaptira prema širini SWAR reči i broju traka, ali nema novih instrukcija za različite širine reči u programu!

SWAR/Polimorfizam (2)

- *Sequencer* prosleđuje grupama funkcionalnih jedinica neophodne kontrolne informacije koje su izvedene iz koda koji se vektorizuje i dinamičke konfiguracije
- Izračunavanje rezultata operacije nad prosleđenim argumentima se u funkcionalnim jedinicama odvija na rekonfigurabilnom hardveru koji podržava izračunavanja, bez obzira na širinu pojedinačnih elemenata niza koji čine argumente

Predikatski registarski fajl

- Može da čuva prethodne predikate, a PLU pravi logičke operacije prethodnih predikata sa novoizračunatim u PLU, **po pojedinačnim elementima niza**, ali istovremeno, sa više bita za sve elemente u SWAR reči.



Primer transformacije algoritma

```
sum:=0
do i=0,1023
    sum:= sum + a(i)                    a(i) je integer
```

Kako paralelizovati na Hwacha, ako je a(i) širine 32 bita, a 4 trake sa po 128 bita?

4*4*4 u 16 SWAR registara, po jedan u svakoj banci.

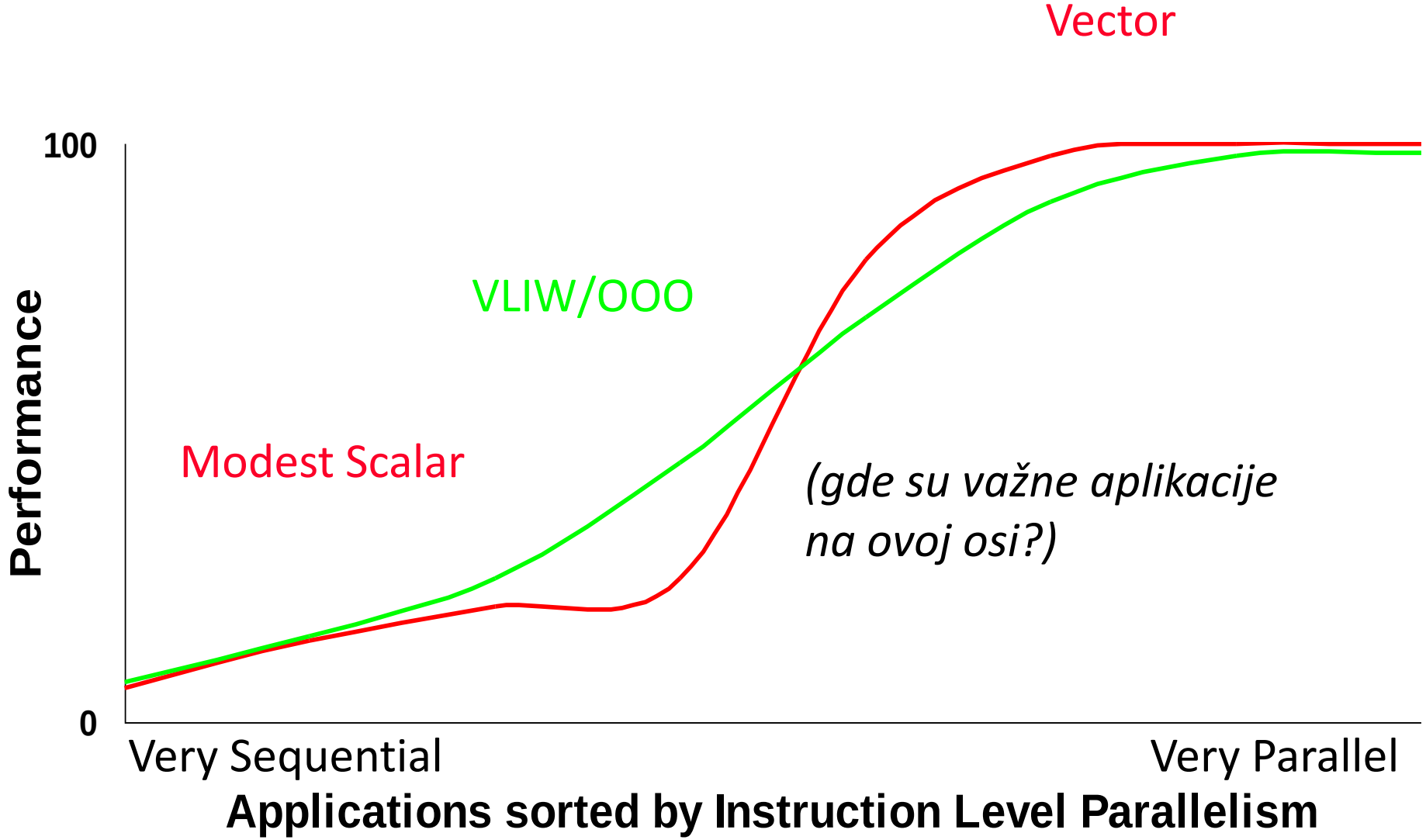
Reči u SWAR registru * broj banki registara u traci *broj traka * broj SWAR registara u svakoj banci

Sabrati podzbirove po bankama u podzbir trake od po 64 * a(i) *4 u jedan SWAR registar

Skupiti u jednu traku i sabrati SWAR podzbirove svih traka 256 * a(i) *4 u SWAR cele petlje

Vratiti SWAR registar u data cache. Sabrati 4 susedna elementa koji su bili u SWAR registru

VLIW/Out-of-Order prema Modest Scalar+Vector



Zaključci

- Potrebno je praviti algoritme koji sadrže petlje koje je moguće vektorizovati
- To mora da uradi programer, jer programski prevodilac ne može da ulazi u algoritam
- Novi programski prevodioci će takve programe moći da pretoče u veoma paralelni kod za vektorske procesore. Zasada vektorske instrukcije.
- Postići će se znatno veća brzina i istovremeno manja potrošnja energije

Vektorski procesori klasični - mane

- Posvećeno suviše pažnje DoAll, ignorišući pripremu za vektorske operacije i podrazumevajući veliki broj iteracija (pipeline je sa velikim brojem pipeline stepeni pa traži puno iteracija da se isplati) !
- Povećanje vektorskih performansi bez posvećivanja dovoljno pažnje skalarnim performansama (Amdahl-ov zakon)
- Nedovoljna je propusnost memorije u odnosu na performanse vektorskog procesora

Prednosti vektorskih procesora

- Lako se dobijaju visoke performanse; N operacija se simultano izvršava za DoAll petlje:
- Skalabilno (dobijaju se bolje performanse sa porastom HW resursa)
- Kompaktno: N operacija se prikazuje jednom jednostavnom instrukcijom (nasuprot VLIW)
- Prediktabilne (real-time) performanse prema statističkim performansama (cache)
- Multimedia, spremno i može se birati N * 64b, 2N * 32b, 4N * 16b, 8N * 8b
- Za chaining je razvijena tehnologija compiler-a
- Za SIMD je još u razvoju tehnologija kompajlera