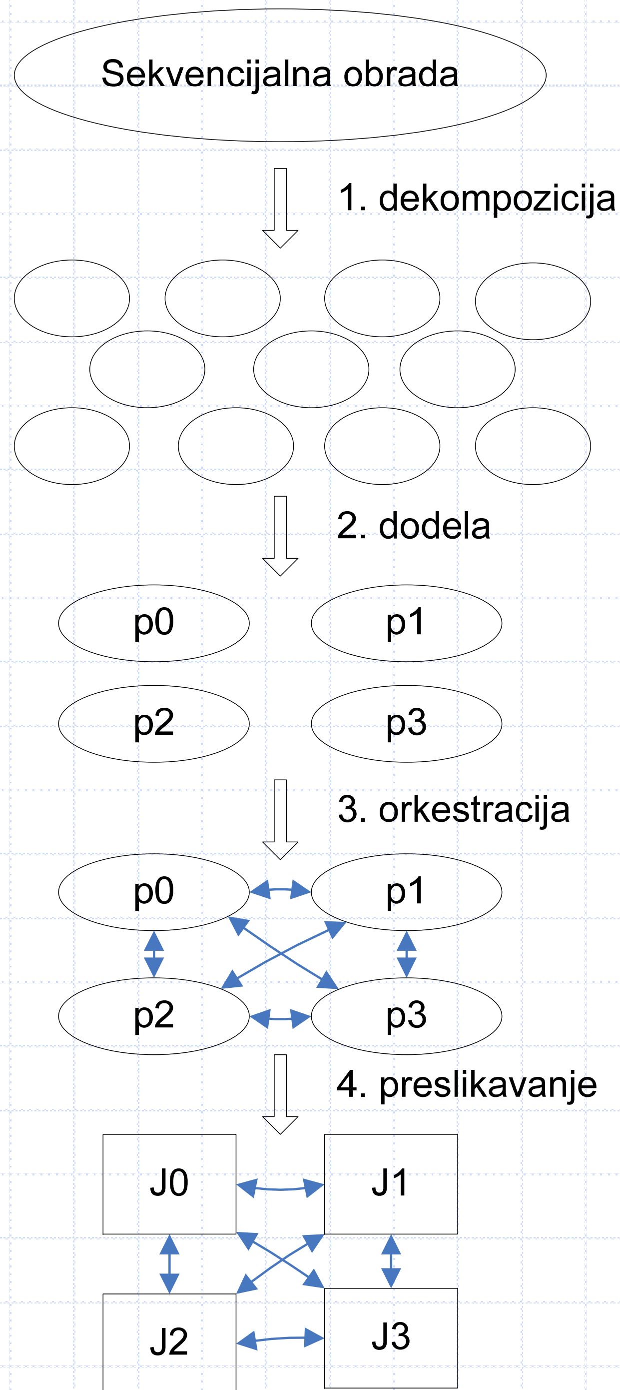
Paralelno programiranje I

- Projektantski šabloni
- Otkrivanje paralelizma

4 Koraka paralelizacije programa



Dekompozicija (Amdahalov zakon)

- Pronaći paralelizam i odlučiti se za nivo njegove upotrebe
- Razbiti obradu na zadatke, koji će biti podeljeni između procesa
  - Zadaci mogu nastajati dinamički
  - Br. zadataka može varirati u vremenu
- Dovoljno zadataka koji će uposliti procesore
  - Br. raspoloživih zadataka u određenom trenutku predstavlja gornju granicu ostvarivog ubrzanja

Dodela (Granularnost)

- Odrediti mehanizam podele posla na jezgara
  - Balanisran posao i smanjena komunikacija
- Po mogućnosti koristiti struktuirane pristupe
  - Inspekcija koda ili razumevanje aplikacije
  - Dobro poznati projektantski šabloni
- Najpre treba raditi na particionisanju obrade
  - Nezavisno od elemenata fizičke arhitekture i modela programiranja
  - Složenost (complexity) obično utiče na odluke

Orkestracija i Mapiranje (Lokalnost)

- Preklapanje obrade i komunikacije
- Očuvati lokalnost podataka
- Raspoređivati zadatke tako da se na vreme (ranije) zadovolje zavisnosti između podataka

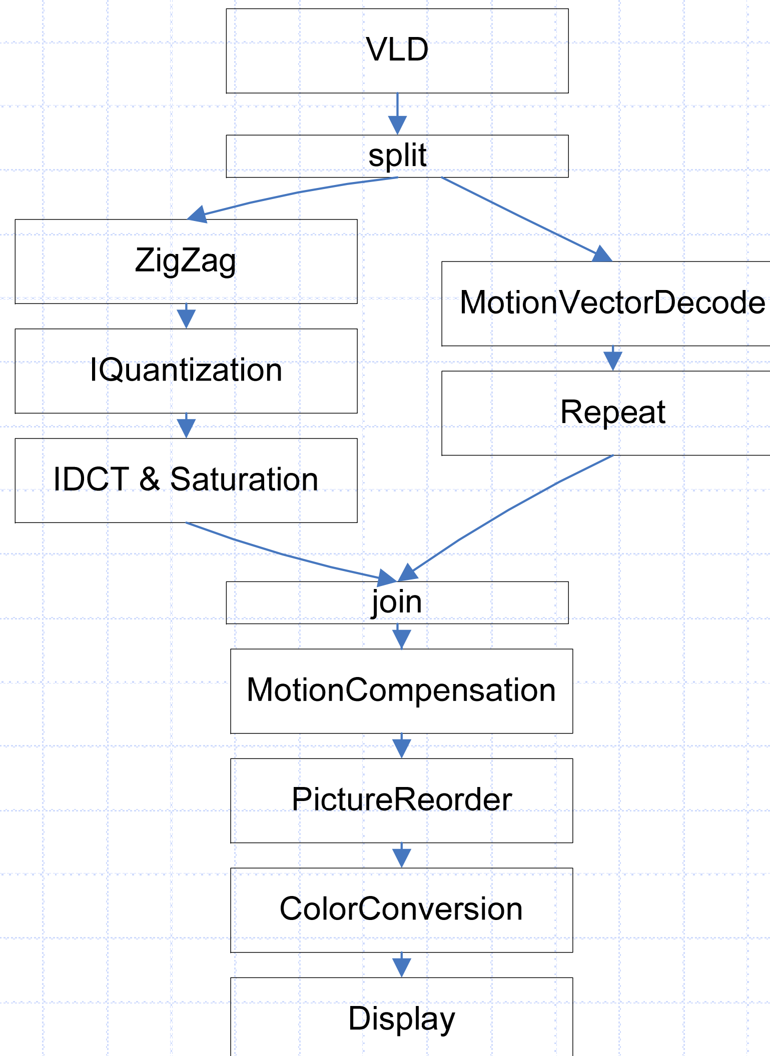
Šabloni paralelnog programiranja

- Knjiga Patterns for parallel programming, autori Mattson, Sanders i Massingill (2005)
  - Recepti za sistematično vođenje programera
- Obezbeđuje rečnik za komunu programera
  - Svaki šablon ima ime – olakšava diskusiju rešenja
- Doprinosi ponovnoj upotrebi i modularnosti
  - Šabloni su pisani u propisanom formatu
  - Čitalac brzo razume rešenje i njegov kontekst
- Inače, suviše teško za programere i ne iskorišćava potpuno paralelnu arhitekturu

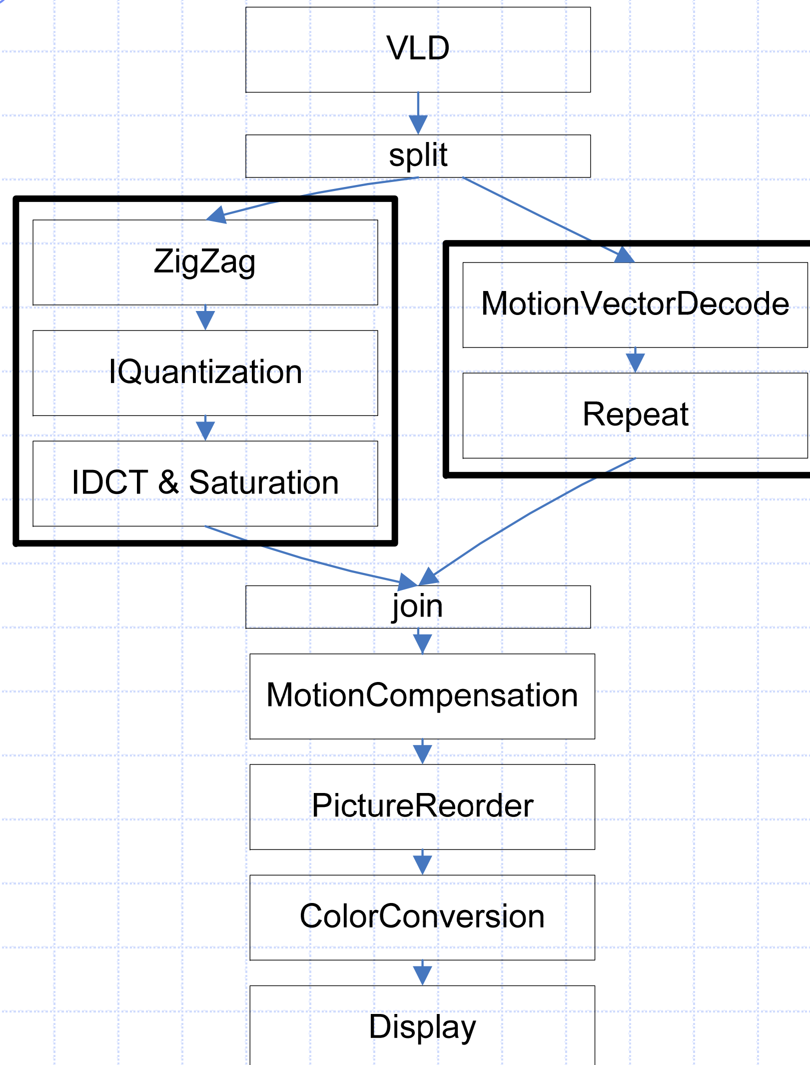
Četiri projektantska prostora

- Izražavanje algoritma
  - I Pronalaženje paralelizma
    - Izlaganje konkurentnih zadataka
  - II Struktura Algoritma
    - Preslikavanje zadataka na procese radi korišćenja paralelnih arhitektura
- Konstruisanje programa
  - III Pomoćne strukture
    - Šabloni koda i struktura podataka
  - IV Izvedbeni mehanizmi
    - Mehanizmi niskog nivoa, koji se koriste za pisanje paralelnih programa

Evo algoritma.
Gde je paralelizam?

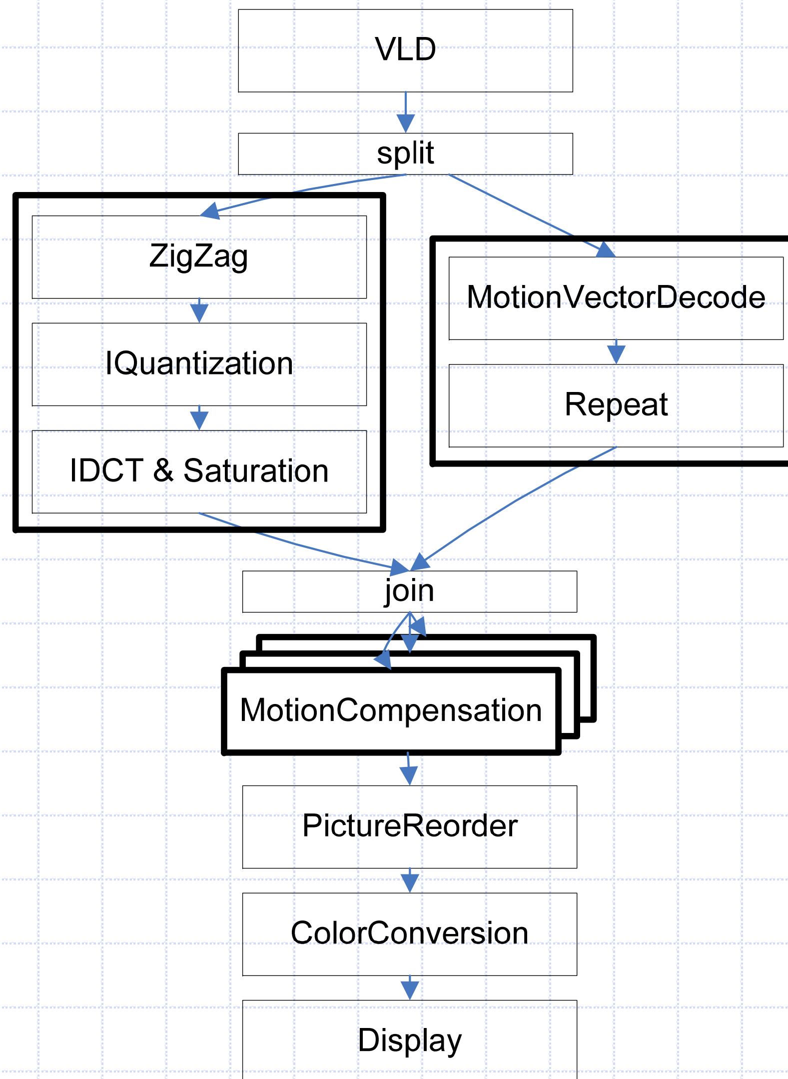


Evo algoritma. Gde je paralelizam?



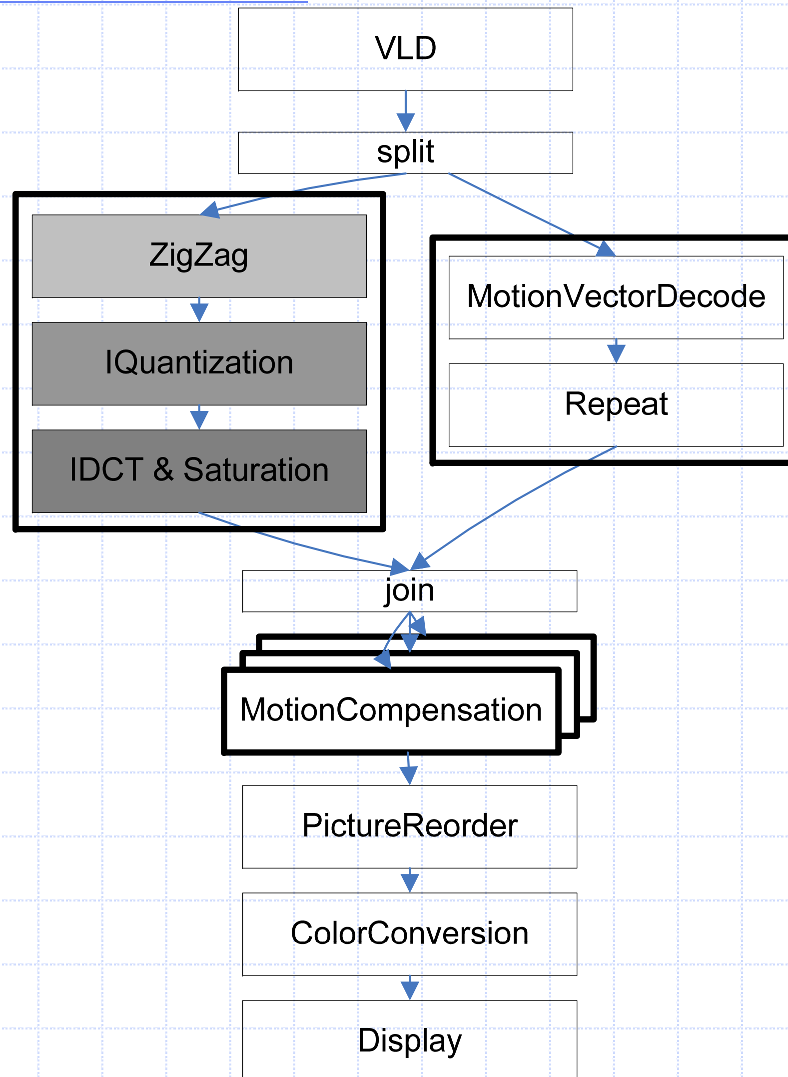
- Dekompozicija zadataka
  - Nezavsine grube obrade
  - Svojstvene algoritmu
- Niz iskaza koji operišu kao grupa
  - Odgovaraju nekom logičkom delu programa
  - Obično prate način na koji programer razmišlja o problemu

Evo algoritma. Gde je paralelizam?



- Dekompozicija zadataka
  - Paralelizam u aplikaciji
- Dekompozicija podataka
  - Ista obrada se primenjuje na malim blokovima podatka, koji su izvedeni iz velikog skupa podataka

Evo algoritma. Gde je paralelizam?



- Dekompozicija zadataka
  - Paralelizam u aplikaciji
- Dekompozicija podataka
  - Ista obrada puno podataka
- Dekompozicija protočne obrade
  - Linije sklapanja podataka
  - Lanci proizvođača-potrošača

Uputstvo za dekompoziciju zadataka (1/2)

- Algoritmi nastaju iz dobrog razumevanja problema, koji se rešava
- Programi se obično prirodno dekomponuju na zadatke
  - Dve česte dekompozicije su:
    - Pozivi funkcija
    - Različite iteracije u petlji
- Lakše je započeti sa više zadataka i kasnije ih spajati, nego sa manje zadataka, koje kasnije treba razdvajati

Uputstvo za dekompoziciju zadataka (2/2)

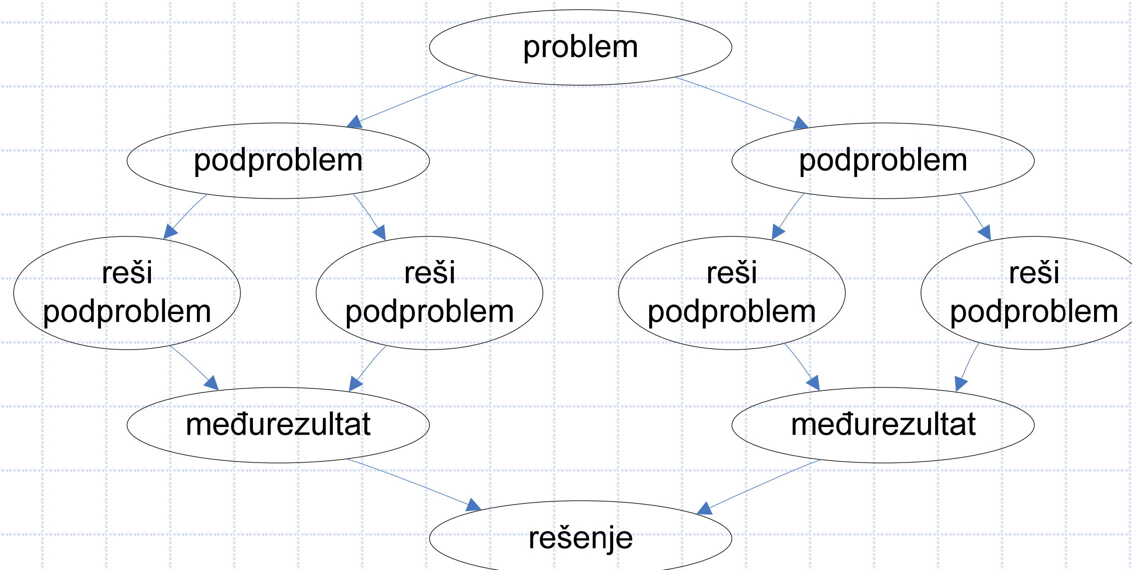
- Fleksibilnost u broju i veličini zadataka
  - Ne treba da budu vezani za specifičnu arhitekturu
  - Fiksni zadaci naspram parametrizovanih zadataka
- Efikasnost
  - Zadaci sa dovoljno posla da se amortizuje cena njihovog stvaranja i rukovanja
  - Zadaci treba da budu dovoljno nezavisni, kako rukovanje zavisnostima ne bi bilo usko grlo
- Jednostavnost
  - Kod mora ostati čitljiv i lak za održavanje/praćenje

Uputstvo za dekompoziciju podataka

- Dekompozicija podataka je obično određena dekompozicijom zadataka
- Programer mora adresirati ove dve dekompozicije da bi napravio paralelni program
  - Sa kojom započeti?
- Dekompzicija podataka je dobra početna tačka kada je:
  - Glavna obrada organizovana oko manipulacije velike strukture podataka
  - Primenjuju se slične operacije nad različitim delovima strukture podataka

Česte dekompozicije podataka

- Strukture podataka u obliku nizova
  - Dekompozicija nizova po vrstama, kolonama, blokovima
- Rekurzivne strukture podataka
  - Na primer: dekompozicija stabla u podstabla



Uputstvo za dekompoziciju podataka

- Fleksibilnost
  - Broj i veličina blokova podataka treba da podrže širok opseg izvršenja
- Efikasnost
  - Blokovi podataka treba da generišu uporedive količine posla (radi balansiranja opterećenja)
- Jednostavnost
  - Složene kompozicije podataka mogu biti teške za obradu i otkrivanje grešaka (debug)

Slučaj za dekompoziciju protočne obrade

- Podaci teku kroz niz stepeni obrade
  - Dobra analogija je linija za sklapanje proizvoda
- Glavni primer u računarskim arhitekturama?
  - Protočna obrada instrukcija u modernim CPU
- Primer u komandnom procesoru OS Unix?
  - Cevi (pipes) u Unix: cat foobar.c | grep bar | wc
- Drugi primeri
  - Obrada signala
  - Grafika

Ponovni inženjering radi paralelizacije programa (1/3)

- Najčešće se kreće od sekvencijalnih programa
  - Lakše za pisanje i otkrivanje grešaka
  - Nasleđeni (legacy) kod
- Kako pretvoriti sekvencijalan program u paralelni
  - Izučiti teren
  - Šabloni nude pitanja za procenu postojećeg koda
  - Većina pitanja su ista kao u bilo kom inženjeringu
  - Da li se program numerički dobro ponaša?

Ponovni inženjering radi paralelizacije programa (2/3)

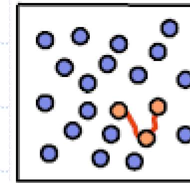
- Definisati predmet rada (scope) i dobiti saglasnost korisnika
  - Zahtevana preciznost rezultata
  - Opseg ulaznih podataka
  - Očekivanja u pogledu performanse
  - Izvodljivost (proračun na "zadnjoj strani koverte")
- Definisati protokol testiranja/prijema radova

Ponovni inženjering radi paralelizacije programa (3/3)

- Odrediti vruće tačke: Gde se troši najviše vremena?
  - Analiza uvidom (gledanjem) u kod
  - Analiza pomoću alata za profilisanje koda
- Paralelizacija
  - Krenuti od vrućih tačaka
  - Napraviti niz malih izmena, svaka praćena testiranjem
  - Šabloni obezbeđuju vođenje kroz ovaj proces

Primer: dinamika molekula

- Simulirati kretanje u molekularnom sistemu
  - Npr. radi razumevanja interakcije lekova i proteina
- Sile
  - Ograničene sile unutar molekula
  - Sile dalekog dometa između atoma



- Naivni algoritam ima nxn interakcija: nije izvodiv
- Koristi metod odsecanja: posmatraj samo sile između suseda koji su dovoljno blizu

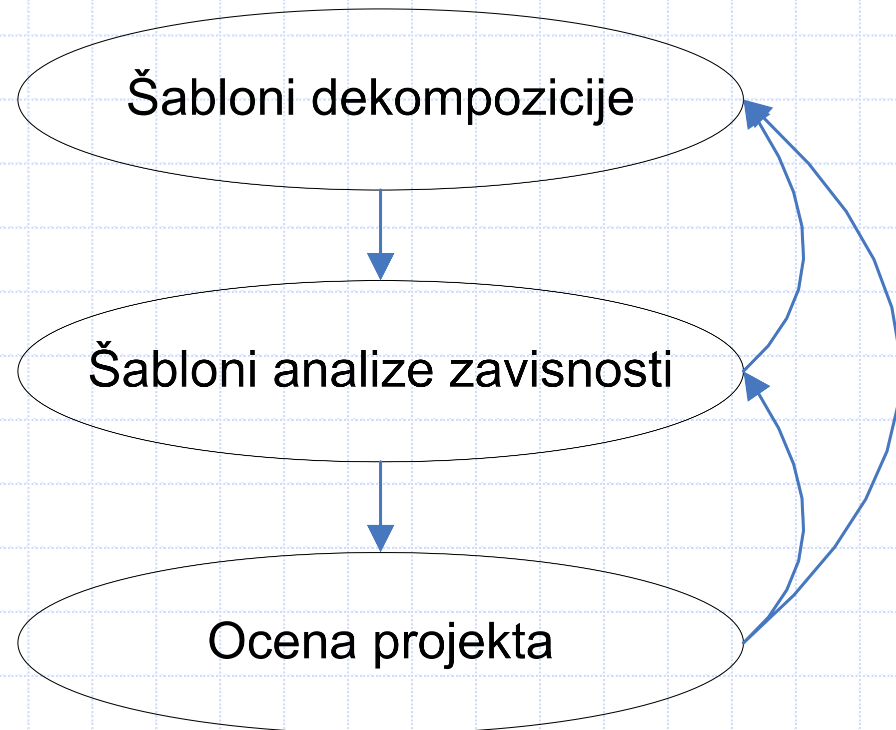
Simulator molekularne dinamike

```
// pseudo code
real[3,n] atoms
real[3,n] force
int [2,m] neighbors

function simulate(steps)
    for time = 1 to steps and for each atom
        Compute bonded forces
        Compute neighbors
        Compute long-range forces
        Update position
    end loop
end function
```

Projektantski prostor: Pronalaženje paralelizma

- Šabloni dekompozicije
- Šabloni analize zavisnosti
- Evaluacija projekta

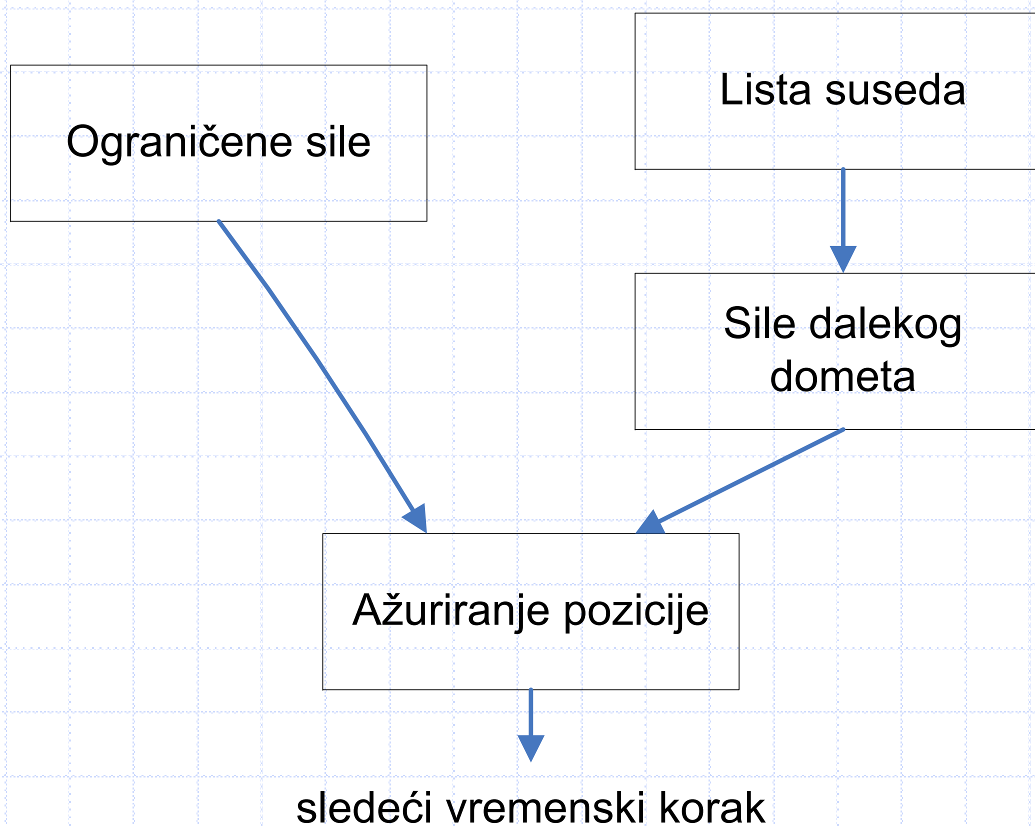


Šabloni dekompozicije

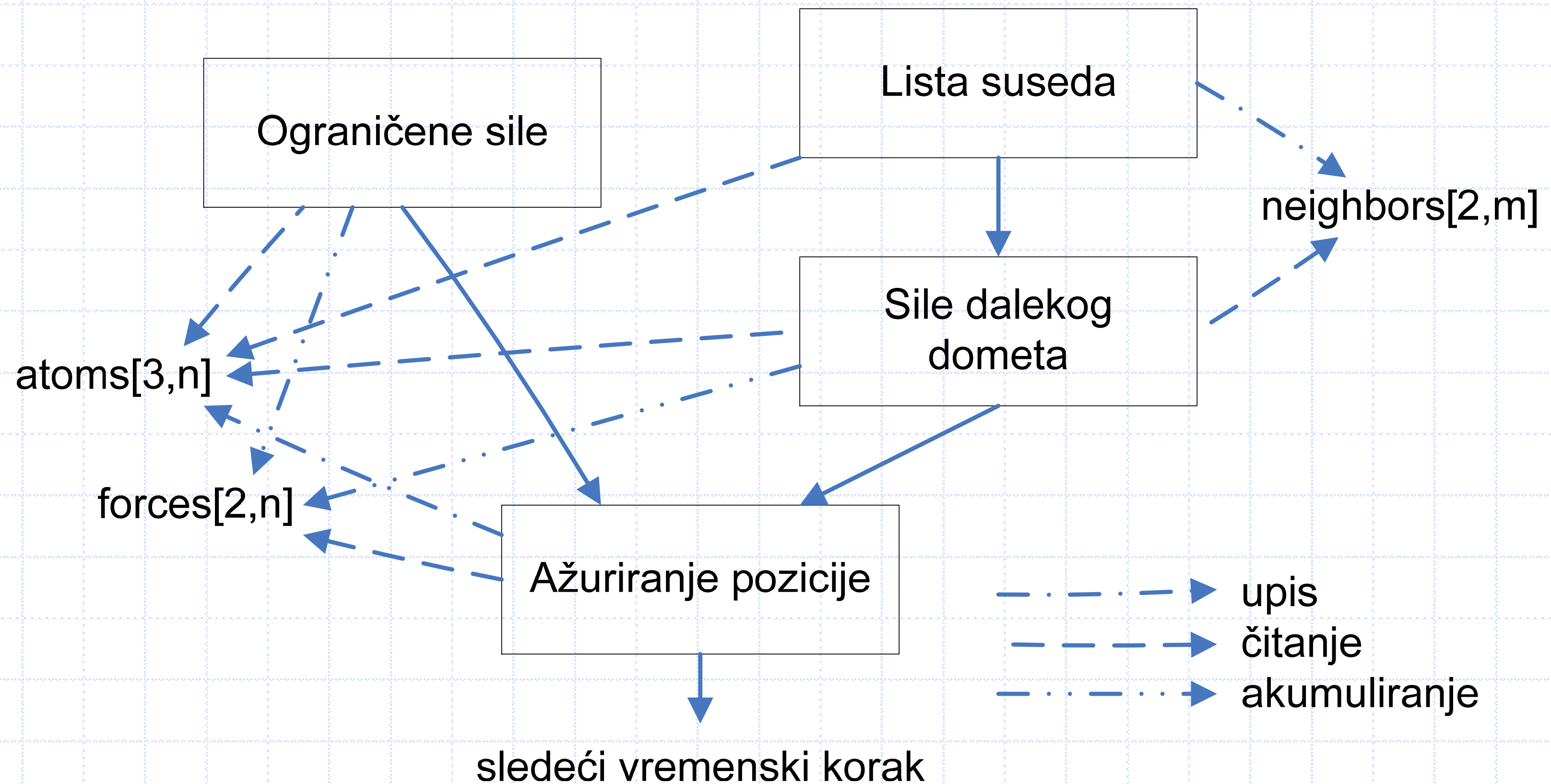
- Glavna obrada je petlja po atomima
- Sugeriše dekompoziciju zadataka
  - Zadatak odgovara jednoj iteraciji petlje
    - Ažuriranje jednog atoma
  - Dodatni zadaci
    - Proračun ograničenih sila
    - Pronalaženje suseda
    - Proračun sila dalekog dometa
    - Ažuriranje pozicije
- Postoje podaci koji su deljeni između atoma

```
for time = 1 to steps and
  for each atom
     Compute bonded forces
     Compute neighbors
     Compute long-range forces
     Update position
end loop
```

Analiza kontrolnih zavisnosti



Analiza zavisnosti podataka



Evaluacija projekta

- Koja je ciljna arhitektura?
  - Deljena memorija, distribuirana memorija, prosleđivanje poruka
- Da li podaci imaju takve prostorne osobine (samo očitavanje, akumuliranje, vremenska ograničenja) da se njihovim zavisnostima može baratati efikasno?
- Ako projekat zadovoljava, pređi na sledeći projektantski prostor