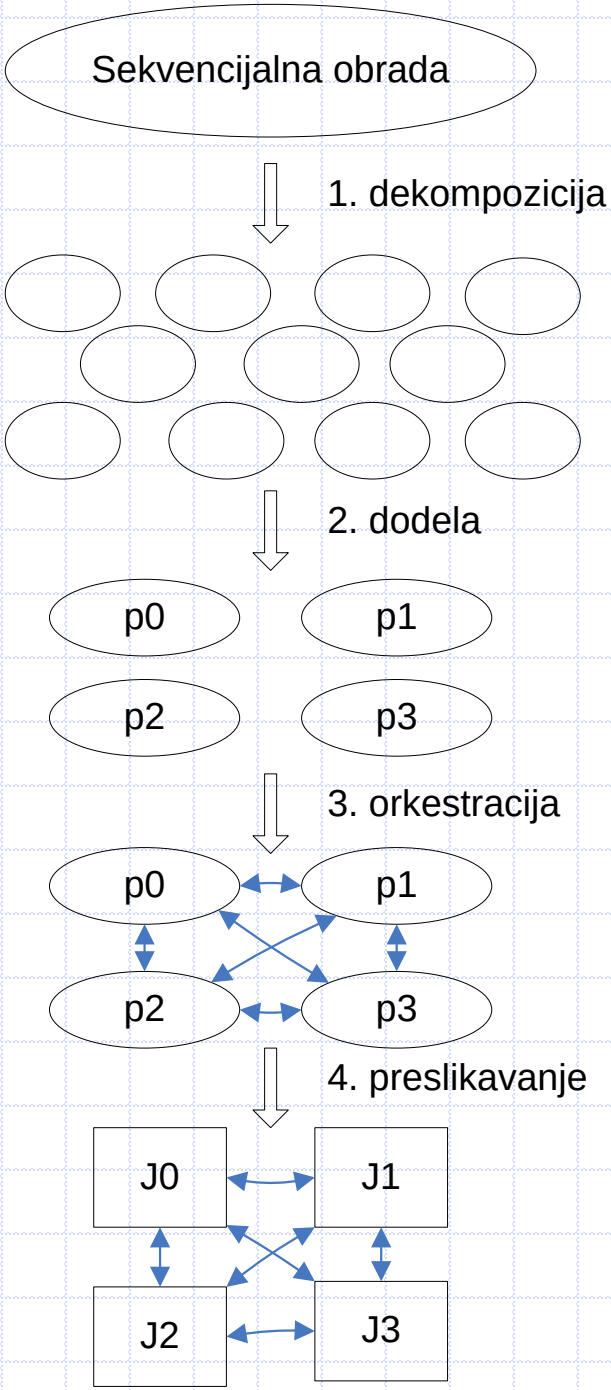
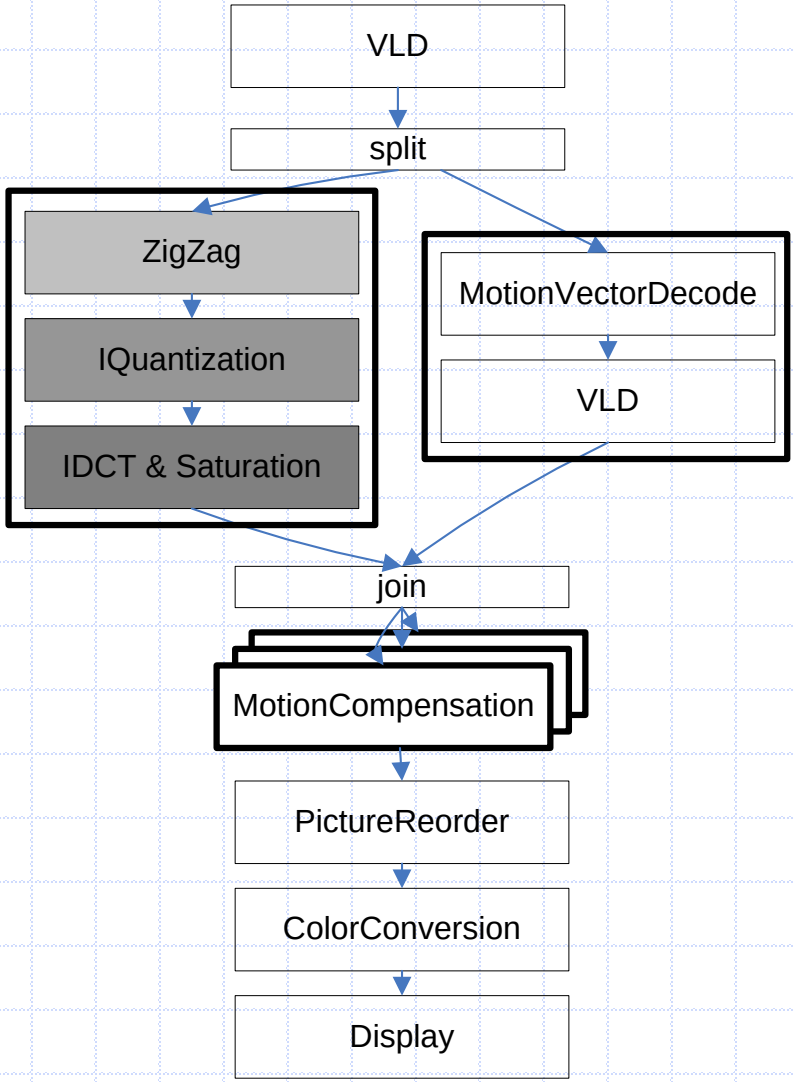
Paralelno programiranje II

- Analiza zavisnosti
- Struktura algoritma
- Podržavajuće strukture
- Komunikacioni šabloni

4 Koraka paralelizacije programa



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

Analiza zavisnosti

- Ako su data dva zadatka, kako odrediti da li se oni mogu sigurno/ispravno izvršiti paralelno?

Bernštajnov uslov

- Ri skup mem. lokacija koje čita zadatak Ti (ulaz)
- Wj skup lokacija u koje piše zadatak Tj (izlaz)

- Dva zadatka, T1 i T2, mogu biti paralelna ako:
  - Ulaz T1 nije deo izlaza T2
  - Ulaz T2 nije deo izlaza T1
  - Izlazi T1 i T2 se ne preklapaju

Primer: Da li ova dva zadatka mogu biti paralelna?

T1	T2
a = x + y	b = x + z

$R_1=\{x,y\}$, $W_1=\{a\}$, $R_2=\{x,z\}$, $W_2=\{b\}$

$R_1 \cap W_2 = \{x,y\} \cap \{b\} = \emptyset$

$R_2 \cap W_1 = \{x,z\} \cap \{a\} = \emptyset$

$W_1 \cap W_2 = \{b\} \cap \{a\} = \emptyset$

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

Projektantski prostor: Struktura algoritma

- Sledeći korak je preslikavanje zadataka na jedinice izvršenja (procesi/niti)

- Važno je razmotriti
  - Broj jedinica izvršenja koje podržava ciljna platforma
  - Cenu deljenja informacije između jedinica izvršenja
  - Izbegavati suvišno ograničavanje implementacije
    - Radi dobro na ciljnoj platformi
    - Dovoljno fleksibilna da se lako adaptira za različite arhitekture

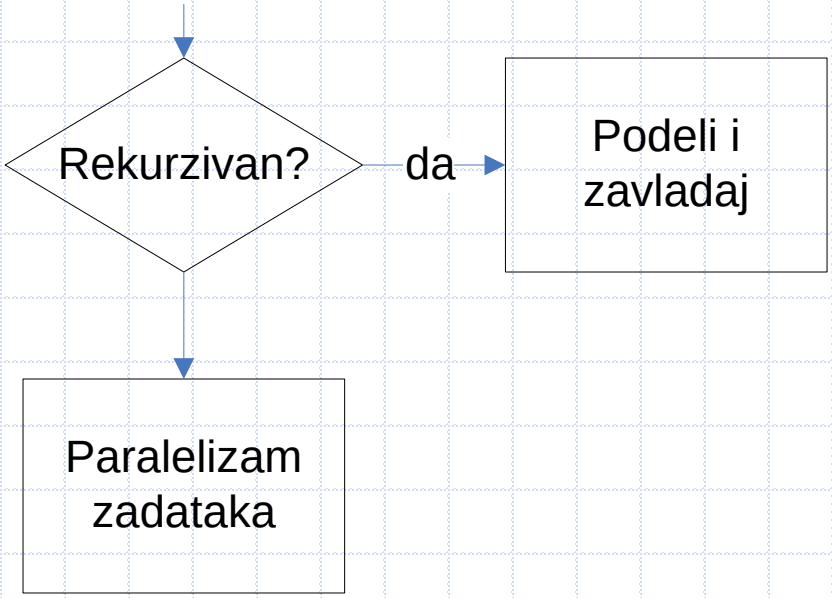
Glavni princip organizovanja

- Kako odrediti strukturu algoritma, koja predstavlja preslikavanje zadataka na jedinice izvršenja?

- Paralelizam obično određuje glavni princip organizovanja
  - Organizovanje po zadacima
  - Organizovanje po dekompoziciji podataka
  - Organizovanje po toku podataka

Organizacija po zadacima

- Rekurzivni program => podeli i zavladaj
- Nerekurzivni program => paralelizam zadataka

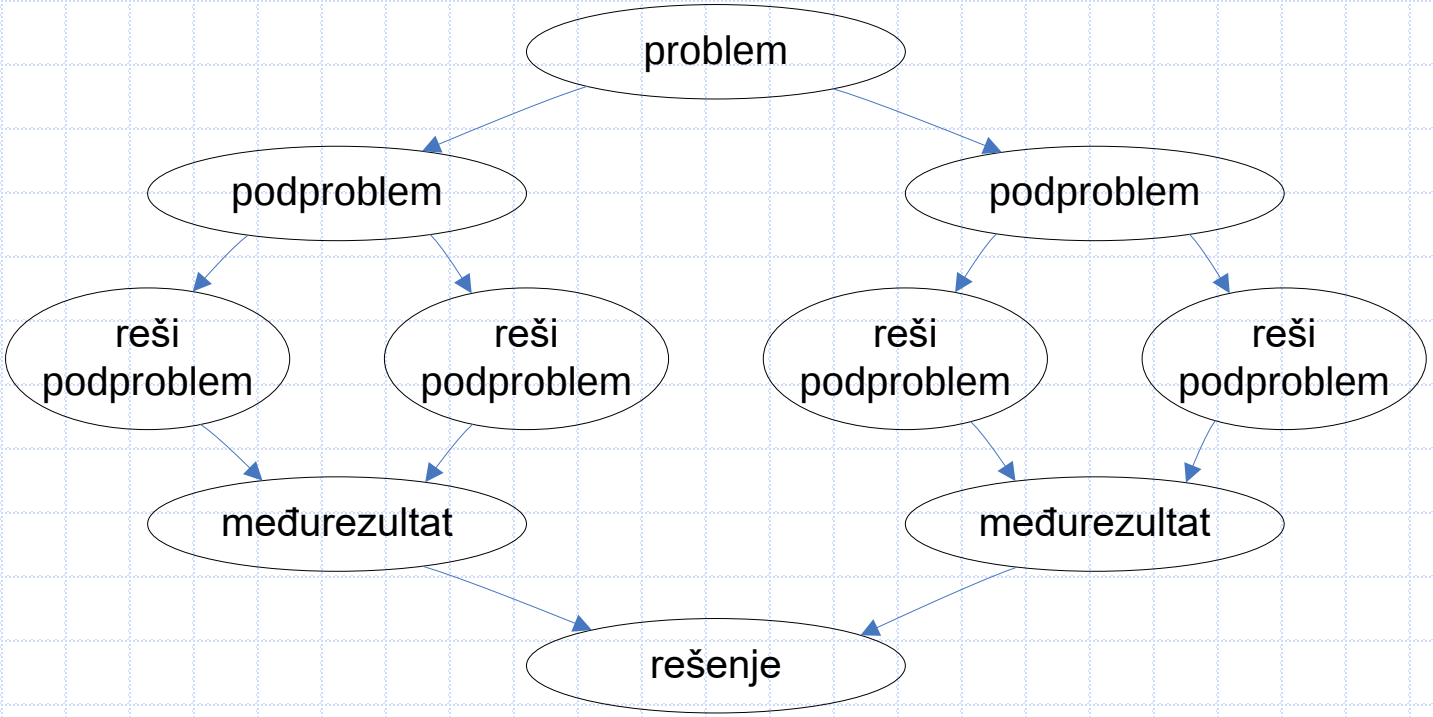


Paralelizam zadataka

- Praćenje zraka (ray tracing)
  - Obrada za svaki zrak je odvojena i nezavisna
- Dinamika molekula
  - Obrada neograničenih sila, neke zavisnosti
- Zajednički faktori
  - Zadaci su pridruženi iteracijama petlje
  - Zadaci su uglavnom poznati na početku obrade
  - Svi zadaci se ne moraju završiti da bi se došlo do konačnog rešenja

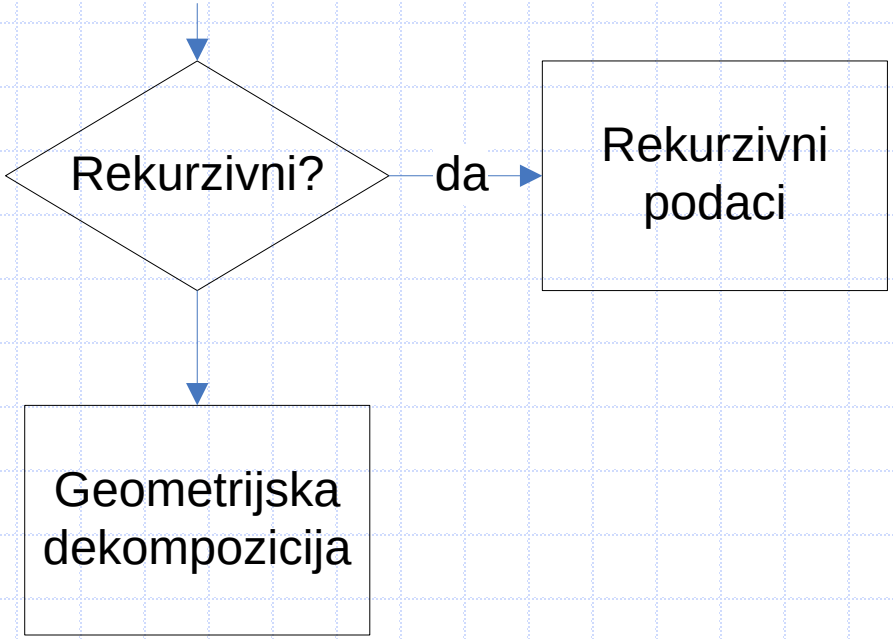
Podeli i zavladaj (divide & conquer)

- Za rekurzivni program: podeli i zavladaj
  - Podproblemi ne moraju biti uniformni (iste veličine)
  - Može zahtevati dinamičko uravnoteženje opterećenja



Organizovanje po podacima

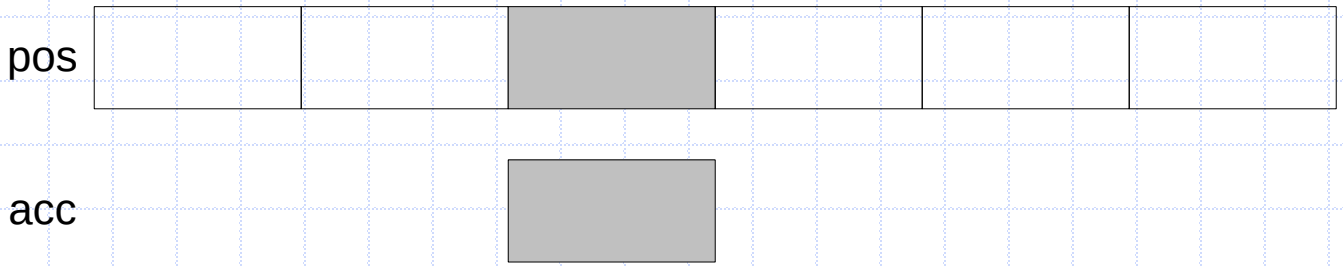
- Operacije na centralnoj strukturi podataka
  - Nizovi i linearne strukture podataka
  - Rekurzivne strukture podataka



Geometrijska dekompozicija

- Simulator gravitacionih tela
  - Izračunati sile između parova tela i ažurirati sile na pojedina tela

```
VEC3D acc[NUM_BODIES] = 0;
for (i = 0; i < NUM_BODIES - 1; i++) {
   for (j = i + 1; j < NUM_BODIES; j++) {
      // Displacement vector
      VEC3D d = pos[j] – pos[i];
      // Force
      t = 1 / sqr(length(d));
      // Components of force along displacement
      d = t * (d / length(d));
      acc[i] += d * mass[j];
      acc[j] += -d * mass[i];
   }
}
```

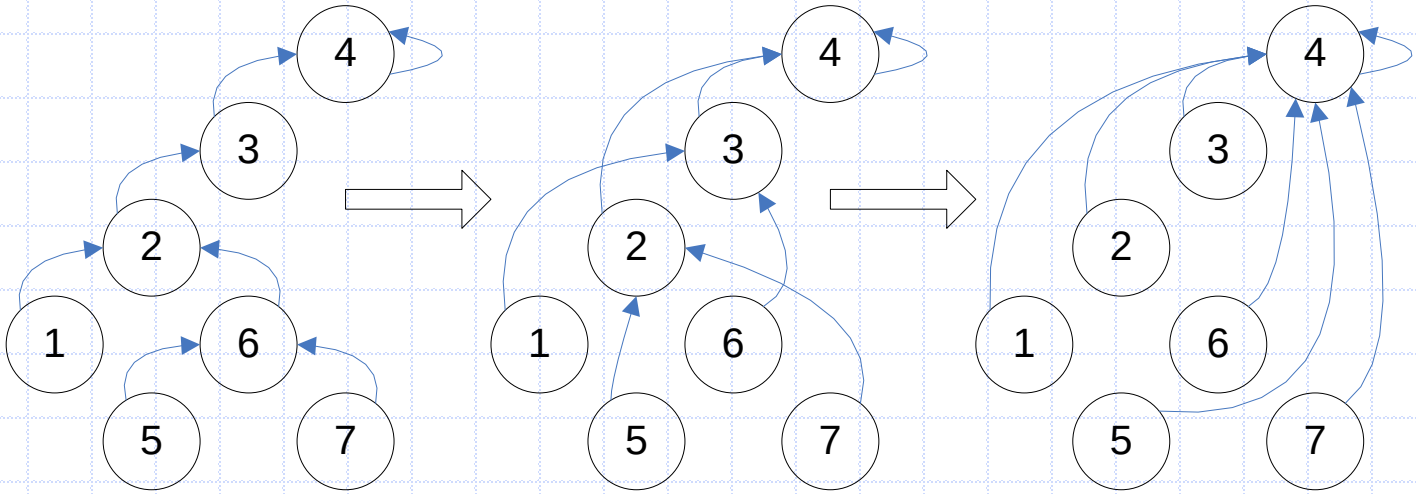


Rekurzivni podaci

- Obrade nad listama, stablima, ili grafovima
  - Obično se pokazuje da je jedini način da se reši problem, da se sekvencijalno ide kroz struktu podataka
- Ali, pojavljuju se i prilike da se operacije preoblikuju tako da se izloži paralelizam

Primer rekurzivnih podataka: Pronalaženje korena

- Ako je data šuma usmerenih stabala sa korenom, pronađi koren stabla koje sadrži čvor
  - Paralelni pristup: za svaki čvor pronađi prethodnikovog prethodnika, ponavljaj dok ima izmena
    - O(log n) naspram O(n)

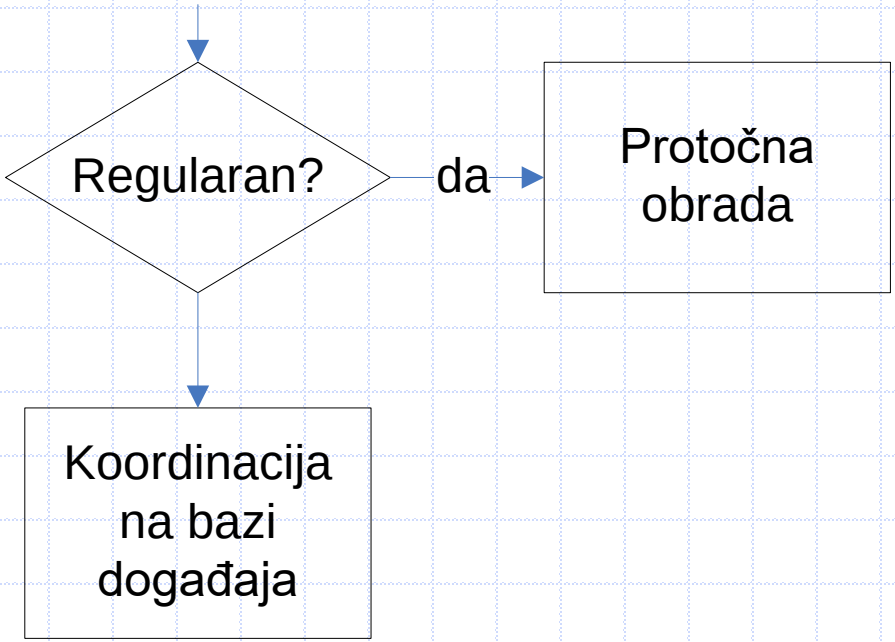


Kompromis: Količina posla naspram paralelizma

- Paralelno prestruktuiranje algoritma za pronalaženje korena dovodi do O(log n) naspram O(n) za sekvencijalan pristup
- Većina strategija zasnovanih na ovoj strategiji dopuštaju povećanje ukupne količine posla radi skraćenja vremena izvršenja na osnovu iskorišćenog paralelizma

Organizacija po toku podataka

- U nekim aplikacionim domenima, tok podataka određuje redosled zadataka
  - Regularan, u jednom smeru, uglavnom stabilan tok
  - Iregularan, dinamički, nepredvidivi tok podataka



Propusnost protočne obrade naspram kašnjenja (latency)

- Paralelizam protočne obrade je ograničen brojem stepeni protočne obrade
- Radi dobro ako je vreme punjenja i pražnjenja protočne obrade malo u odnosu na vreme obrade
- Mera performanse je obično propusnost
  - Stopa kojom se podaci pojavljuju na kraju protočne obrade (npr. broj okvira u sekundi)
- Kašnjenje protočne obrade je važno za aplikacije u realnom vremenu
  - Vremenski interval od ulaza podataka u protočnu obradu do izlaza

Koordinacija zasnovana na događajima

- U ovom šablonu, interakcija zadataka radi obrade podataka se dešava u nepredvidivim intervalima vremena
- U aplikacijama koje koriste ovaj šablon moguća su međusobna blokiranja zadataka (deadlock)

Podržavajuće strukture

- SPMD (Single Program Multiple Data)
- Paralelizam petlje
- Vodeći/Radnik (Master/Worker)
- Grananje/Pridruživanje (Fork/Join)

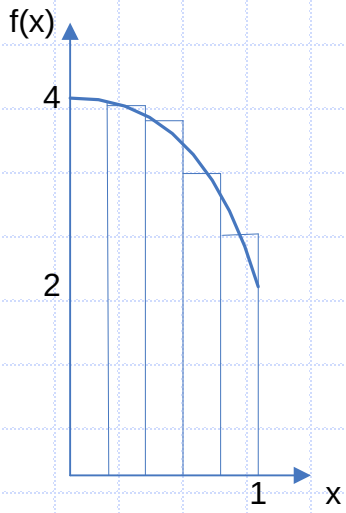
SPMD šablon

- Jedan program više podataka: stvara programe iz jednog izvornog koda, koji se izvršavaju na više procesora
  - Inicijalizacija
  - Dobavljanje jedinstvenog identifikatora
  - Izvršenje istog programa na svakom procesoru
    - Identifikator i ulazni podaci dovode do različitog ponašanja
  - Distribuiranje podataka
  - Dovršavanje (finalizacija)

Primer SPMD: Paralelna numerička integracija

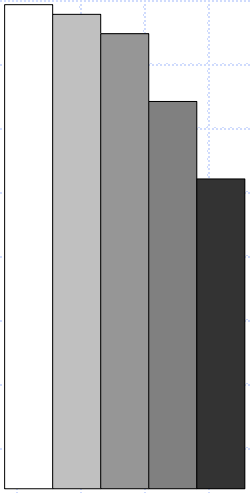
- Računanje broja π numeričkom integracijom funkcije $f(x)=4/(1+x^2)$ nad intervalom (0,1).

```
static long num_steps = 100000;
void main()
{
   int i;
   double pi, x, step, sum = 0.0;
   step = 1.0 / (double) num_steps;
   for (i = 0; i < num_steps; i++){
      x = (i + 0.5) * step;
      sum = sum + 4.0 / (1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n", pi);
}
```



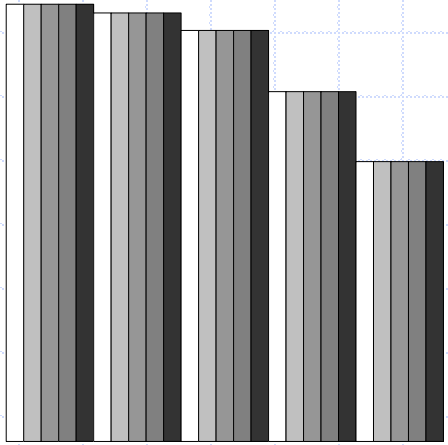
Računanje Pi sa integracijom (MPI)

```
static long num_steps = 100000;
void main(int argc, char* argv[])
{
   int i_start, i_end, i, myid, numprocs;
   double pi, mypi, x, step, sum = 0.0;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
   MPI_BCAST(&num_steps, 1, MPI_INT, 0,
                        MPI_COMM_WORLD);
   i_start = myid * (num_steps/numprocs)
   i_end = i_start + (num_steps/numprocs)
   step = 1.0 / (double) num_steps;
   for (i = i_start; i < i_end; i++) {
      x = (i + 0.5) * step
      sum = sum + 4.0 / (1.0 + x*x);
   }
   mypi = step * sum;
   MPI_REDUCE(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
                           0, MPI_COMM_WORLD);
   if (myid == 0)
      printf("Pi = %f\n", pi);
   MPI_Finalize();
}
```



Distribucija posla: Po bloku naspram Po ciklusu

```
static long num_steps = 100000;
void main(int argc, char* argv[])
{
   int i_start, i_end, i, myid, numprocs;
   double pi, mypi, x, step, sum = 0.0;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
   MPI_BCAST(&num_steps, 1, MPI_INT, 0,
                        MPI_COMM_WORLD);
   i_start = myid * (num_steps/numprocs)
   i_end = i_start + (num_steps/numprocs)
   step = 1.0 / (double) num_steps;
   for (i = myid; i < num_steps; i += numprocs) {
      x = (i + 0.5) * step
      sum = sum + 4.0 / (1.0 + x*x);
   }
   mypi = step * sum;
   MPI_REDUCE(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
                           0, MPI_COMM_WORLD);
   if (myid == 0)
      printf("Pi = %f\n", pi);
   MPI_Finalize();
}
```



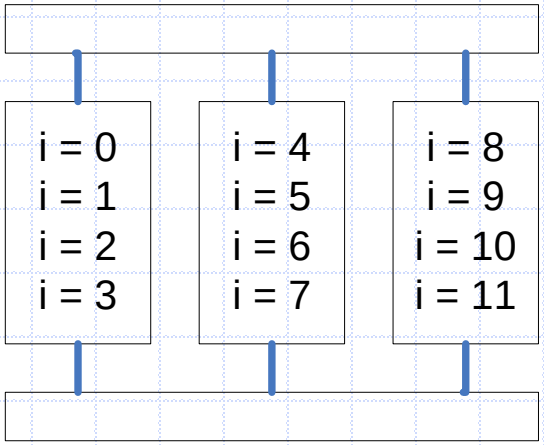
SPMD izazovi

- Razdeliti podatke korektno
- Korektno kombinovati rezultate
- Ostvariti ravnomernu raspodelu posla
- Za programe koji zahtevaju dinamičko uravnoteženje opterećenja, neki drugi šablon je pogodniji

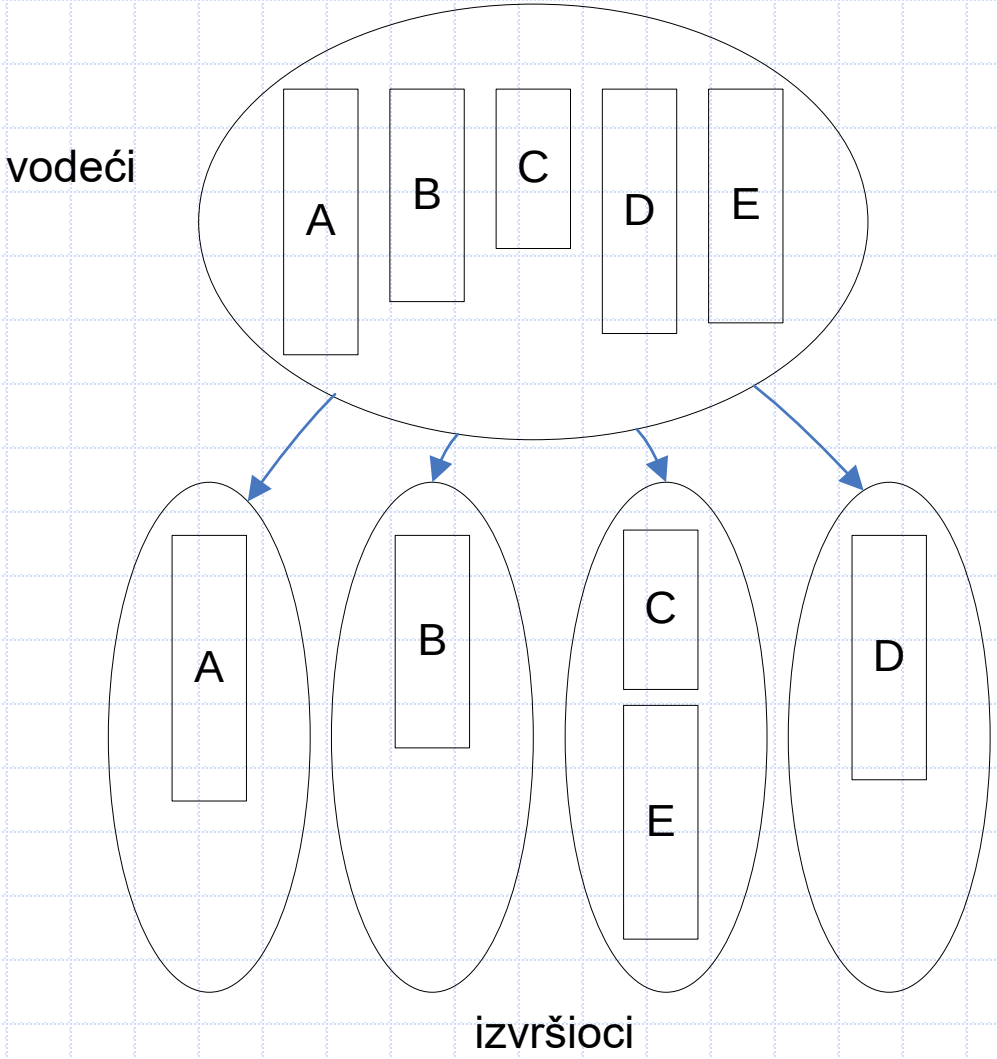
Šablon paralelizma petlje

- Mnogi programi su izraženi korišćenjem iterativnih konstrukcija
  - Modeli programiranja kao što je OpenMP obezbeđuju direktive za automatsku dodelu iteracija petlje pojedinim jedinicama izvršenja
  - Posebno dobar kada se kod ne može masovno prestruktuirati

```
#pragma omp parallel for
for(i = 0; i < 12; i++)
    C[i] = A[i] + B[i];
```



Šablon vodeći/radnik (1/2)



Šablon vodeći/radnik (2/2)

- Posebno relevantan za probleme koji koriste paralelizam zadataka gde zadaci nemaju zavisnosti
  - Zastiđujuće paralelni programi (embarrissingly parallel)

- Glavni izazov je određivanje kada je ceo problem obrađen (postoji potpuno rešenje)

Šablon grananja/pridruživanja

- Zadaci se stvaraju dinamički
  - Zadaci mogu stvoriti još zadataka
- Zadacima se rukuje u skladu sa njihovim odnosima
- Zadatak predak stvara nove zadatke (grananje) zatim čeka da se završe (pridruživanje) pre nego nastavi sa obradom

Komunikacioni šabloni

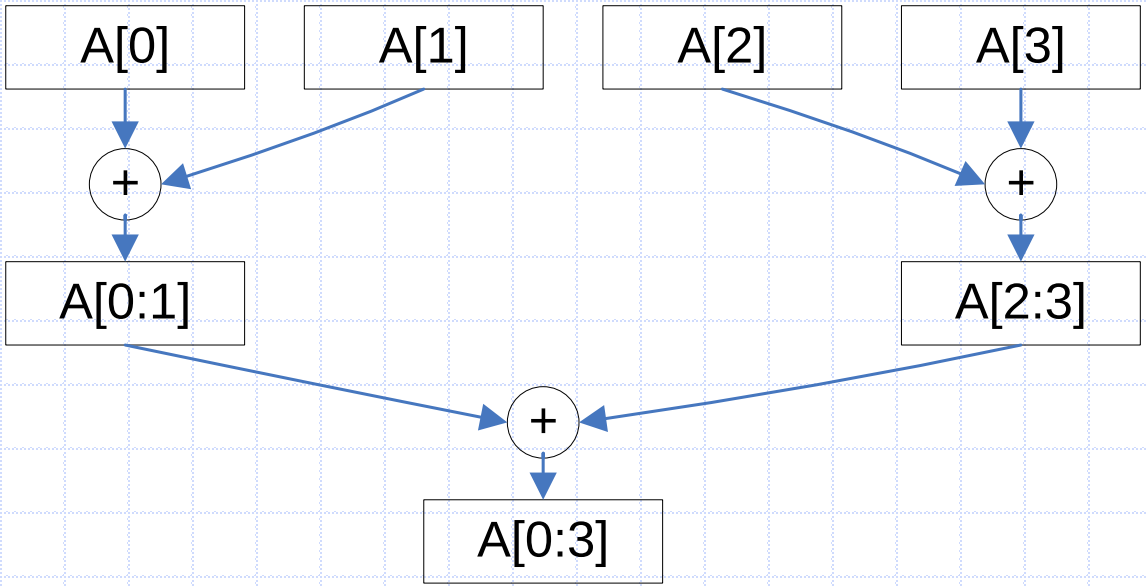
- Tačka-tačka (point-to-point)
- Slanje svima (broadcast)
- Redukcija (reduction)

Serijska redukcija

- Kada operator redukcije nije asocijativan
- Obično ga prati slanje rezultata svima
- Npr. serijsko skupljanje A[0], A[1], A[2], A[3] u čvoru koji ima A[0]:
  - Skupi A[1], dobija se A[0:1]
  - Skupi A[2], dobija se A[0:2]
  - Skupi A[3], dobija se A[0:3]
  - Kraj

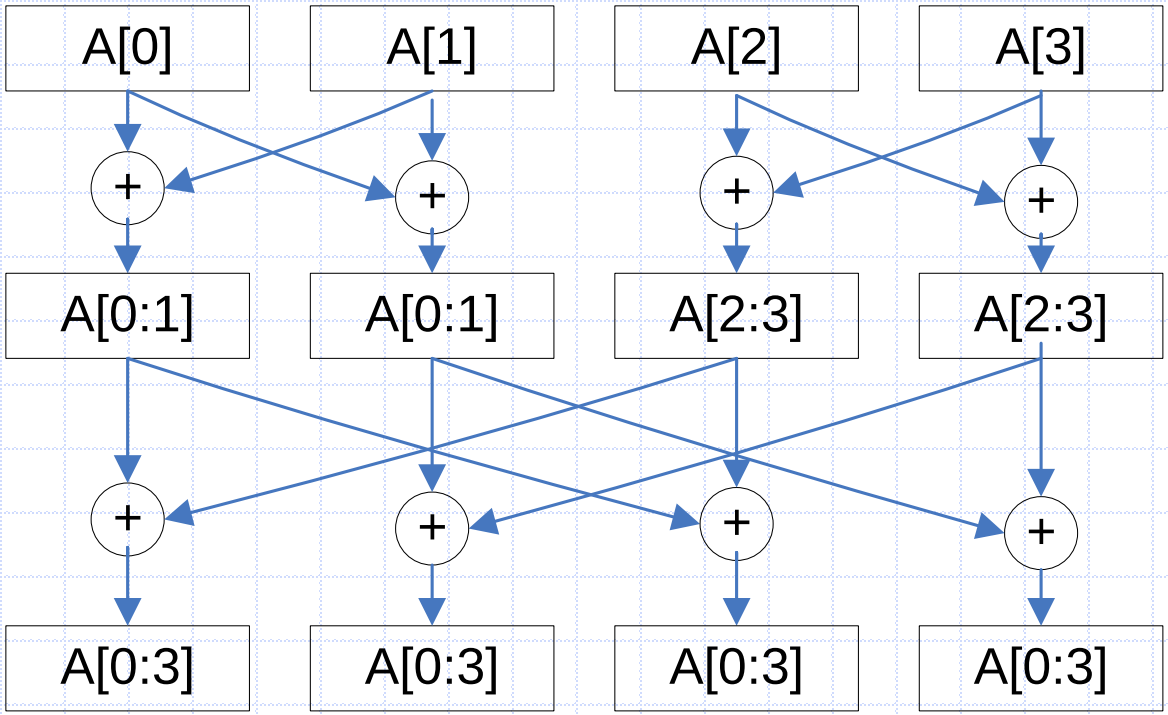
Redukcija zasnovna na stablu

- n koraka za 2^n jedinica izvršenja
- Kada je operator redukcije asocijativan
- Posebno atraktivan ako je rezultat potreban samo jednom zadatku



Rekurzivno-udvajajuća redukcija

- n koraka za 2^n jedinica izvršenja
- Ako svim jedinicama izvršenja treba rezultat redukcije



nastavak... Rekurzivno-udvajajuća redukcija

- Bolja od pristupa zasnovanog na stablu sa slanjem svima (broadcast)
  - Svaka jedinica izvršenja ima kopiju redukovanog rezultata na kraju n koraka
  - U pristupu zasnovanom na stablu sa slanjem svima
    - Redukcija uzima n koraka
    - Slanje svima ne može započeti dok se redukcija ne završi
    - Slanje svima uzima n koraka (zavisno od arhitekture)
    - O(n) naspram O(2n)

Šabloni paralelizacije

- Za kraj sažetak

Struktura algoritma i organizacija

- Šabloni se mogu komponovati hijerarhijski tako da program koristi više od jednog šablona

	Paralelizam zadataka	Podeli i zavladaj	Geometrijska dekompozicija	Rekurzivni podaci	Protočna obrada	Koordinacija na bazi događaja
SPMD	****	***	****	**	***	**
Paralelizam petlje	****	**	***			
Vodeći/ Radnik	****	**	*	*	****	*
Grananje/ pridruživanje	**	****	**		****	****