CRCW PRAM

- načini rješavanja problema istovremenog pisanja:

1. PRIORITETNO
   - samo procesor s najmanjim indeksom uspijeva u pisanju
2. OPERACIJSKI
   - nad svim podacima (koji se upisuju) provodi se zadana operacija (najčešće zbrajanje) i upisuje se rezultat
3. SLUČAJNO
   - jedan slučajno odabrani procesor uspjeva u pisanju (nedeterministički)

⇒ DEFAULT: (3.)

ZAD: Napisati algoritam za CRCW PRAM računalo koji će za zadano polje P ~~izračunati~~ provjeriti ima li u polju elemenata jednakih vrijednosti. Za polje od n procesa dostupno je n procesora. Rezultat mora biti zapisan u jednoj varijabli. Ocijeniti složenost algoritma.

P[] = [1, 2, 3, 2, 4]

```
REZ = ∅;
ZA i=1 DO n-1
    PARALELNO (ZA j=i+1 DO n)
        AKO (P[j] == P[i])
            REZ = 1;   //piše se isti rezultat
```

BROJ KORAKA: n-1
SLOŽENOST: O(n)

⇒ SVAKI USPOREĐUJE JE LI TRENUTNI ELEMENT NIZA JEDNAK NJEGOVOM

```
REZ = ∅;
ZA d=1 DO n/2 + 1
    PARALELNO (ZA j=1 DO n)
        AKO (P[j] == P[j+d])
            REZ = 1;
```

BROJ KORAKA: n/2 + 1
SLOŽENOST: O(n)

⇒ [1 2 3 2 4] → I_1

[1 2 3 2 4] → I_2

[1 2 3 2 4 ...

ZAD: EREW PRAM, dostupna procedura reduciranja uz proizvoljni binarni operator
⇒ provjeriti ima li u polju jednakih vrijednosti, rezultat u jednoj varijabli

```
PARALELNO (ZA j=1 DO n)
    REZ[j] = ∅;
    K[j] = P[j];
ZA d=1 DO n/2 + 1
    PARALELNO (ZA j=1 DO n)
        AKO ( P[j] == K[j+d])
            REZ[j] = 1;
REZ = OR_REDUCE (REZ[]);
```

⇒ nitko ne smije ~~pisati~~ čitati na isto mjesto istovremeno
- naprave se kopije

⇒ nitko istovremeno ne smije pisati na isto mj.
- svaki piše u jednu poziciju polja
- kasnije REDUCE

SLOŽENOST: O(n)

ZAD: Napisati algoritam za CRCW PRAM koji će za zadano polje P odrediti broj različitih elemenata polja.
[1, 2, 1, 3, 4, 2, 5, 1] ⇒ 5

```
REZ = 1;
ZA i=1 DO n-1
    INC = 1;
    PARALELNO (ZA j=i+1 DO n)
        AKO ( P[j] == P[i])
            INC = 0;
    REZ += INC;
```

⇒ elementi se gledaju jedan po jedan i uspoređuju sa svima da li su jednaki
- ako postoji isti → ne broji se
- kad dođemo do tog drugog elementa i iza njega nema više istih ⇒ broji se

⇒ ukupno se broji samo jednom

SLOŽENOST: O(n)

ZAD: U aPRAM na n procesora u lokalnoj memoriji svakog procesora nalazi se podatak VAR. Napiši program kojim se vrijednost podatka svakog procesora postavlja na najveću vrijednost među svim podacima.

ID - indeks procesora
GP[] - globalno polje

GP[ID] = VAR;

ograda

d = 1; ⇒provjera na određenu udaljenost

dok (d<n)

- ako (VAR < GP[ID+d]) → ako je manja moja vrijednost, njegovu upisujem u svoju
  - VAR = GP[ID+d];
- d++;
- ograda

ili ako (VAR > GP[ID+d])

GP[ID+d] = VAR;

⋮

VAR = GP[ID]

ZAD: Napisati algoritam za EREW PRAM koji će za zadano polje P odrediti broj različitih vrijednosti elemenata polja. Rezultat mora biti zapisan u jednoj varijabli.

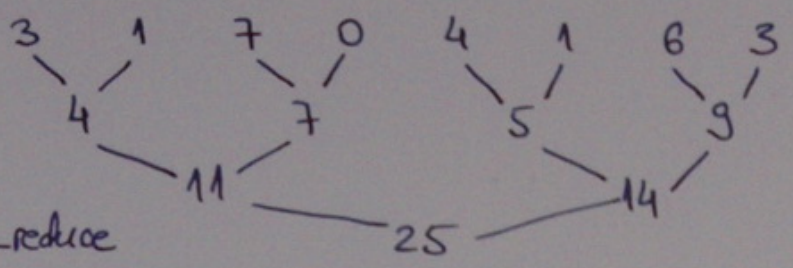
```
PARALELNO (i=1 DO n)
    rez[i] = 1;
    K[i] = P[i];
ZA d=1 DO n/2+1
    PARALELNO (i=1 DO n)
        AKO (K[i] == P[i+d] && rez[i+d] == 1)
            rez[i] = 0;
UKUPNO = 1;
ZA d=1 DO n
    UKUPNO += rez[d]
```

BROJ KORAKA: $1 + n/2 + 1 + n$

SLOŽENOST: $O(n)$

2^i elemenata, 2^i procesora (proizvoljan broj)

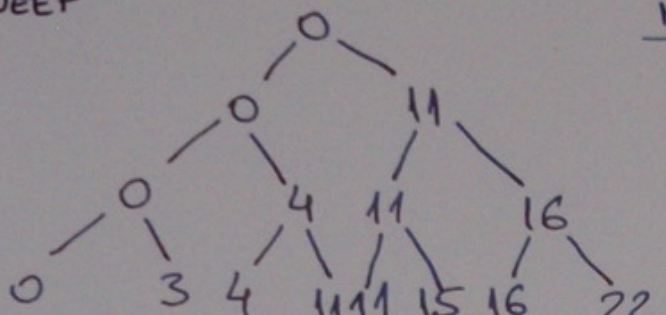
$A[] = [3,1,7,0,4,1,6,3]$, $n = 8$



MEMORIJA

3	1	7	0	4	1	6	3
3	[4]	7	[7]	4	[5]	6	[9]
3	4	7	[11]	4	5	6	[14]
3	4	7	11	4	5	6	[25]

DOWNSWEEP



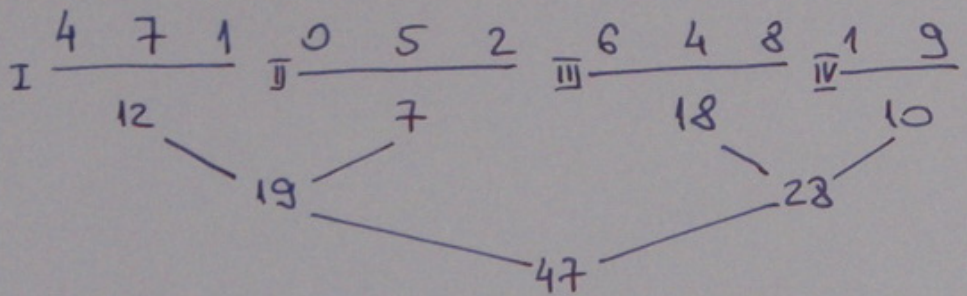
MEMORIJA

3	4	7	11	4	5	6	[0]
3	4	7	[0]	4	5	6	[11]
3	[0]	7	[4]	4	[11]	6	[16]
[0]	[3]	[4]	[11]	[11]	[15]	[16]	[22]

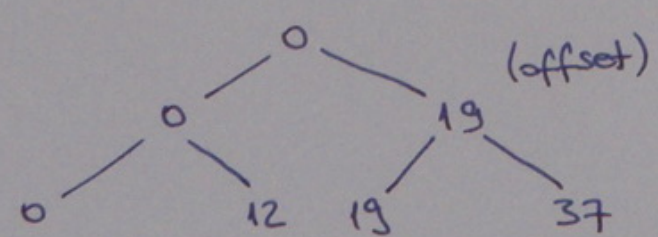
$\neq 2^i$ elemenata, fiksni broj procesora 2^i

$p = 4$, $A = [4,7,1,0,5,2,6,4,8,1,9]$, $n = 11$

$\Rightarrow$ svaki procesor dobiva $\min \lceil n/p \rceil$ elem



DOWNSWEEP



	P#1	P#2	P#3	P#4
POSMAK:	0	12	19	37
	0 4 1	12 12 17	19 25 29	37 38

prescan na 4,7,1

prescan na 6,4,8 → dodaje se +19

$\neq 2^i$ elemenata, proizvoljan broj procesora, +-prescan

$A=[2,3,7,1,4]$, $n=5$

⇒ radimo kao da imamo prvi veći 2^i elemenata,
a te elemente koji fale postavimo na neutralnu vrijednost

2	3	7	1	4	0	0	0
2	[5]	7	[8]	4	[4]	0	[0]
2	5	7	[13]	4	4	0	[4]
2	5	7	13	4	4	0	[17]

DOWNSWEEP

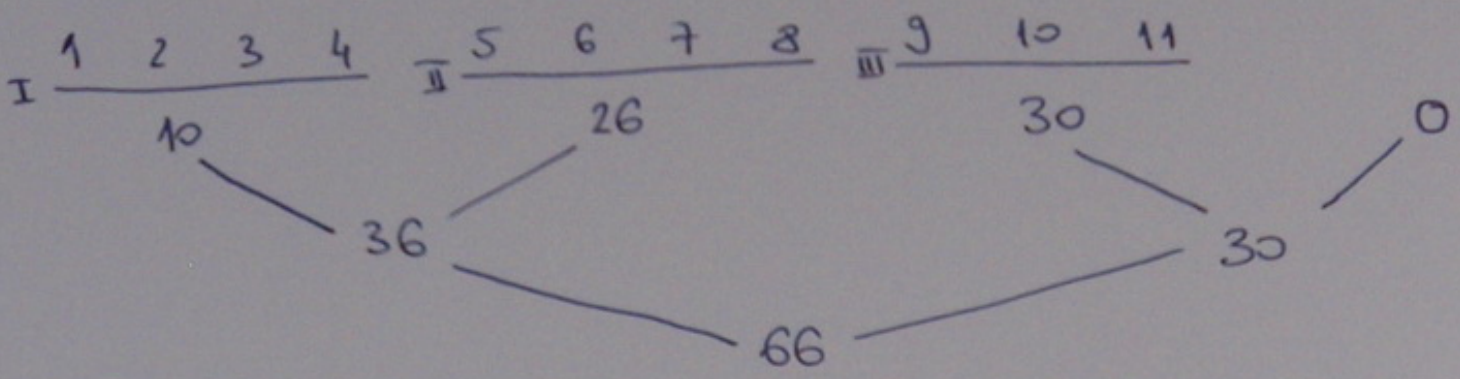
2	5	7	13	4	4	0	[0]
2	5	7	[0]	4	4	0	[13]
2	[0]	7	[5]	4	[13]	0	[17]
[0]	[2]	[5]	[12]	[13]	[17]	[17]	[17]

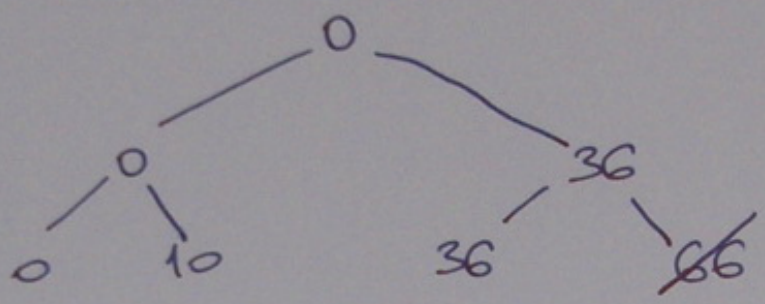
rezultat

$\neq 2^i$ elemenata, p procesora $(p+2^i)$

$A[] = [1,2,3,4,5,6,7,8,9,10,11]$, $n = 11$

$p = 3$ $\quad \lfloor n/p \rfloor = 3$ (minimalno)





POSMAK:

P#1	P#2	P#3
0	10	36

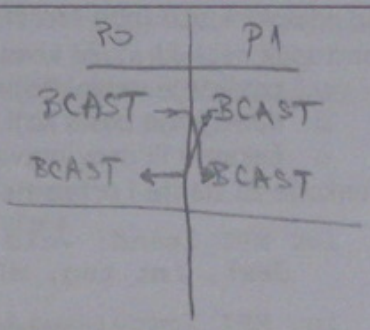
0 1 3 6 | 10 15 21 28 | 36 45 55

prescan na 1,2,3,4

2.6 Ispravnost programa

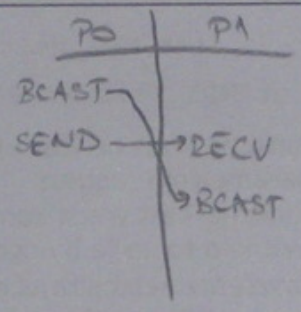
- *Pitanje*: uključuju li funkcije globalne komunikacije međusobnu sinkronizaciju procesa?
- *Odgovor*: Ne! standardom nije propisano hoće li se procesi prilikom globalne kom. sinkronizirati (kao sporedni učinak) ili ne, već to ovisi o izvedbi navedenih funkcija
- u većini implementacija globalne komunikacije izvedene su kao niz *point-to-point* funkcija, pa sinkronizacija ovisi i o trenutnim uvjetima rada (veličina poruke i sl.)
- Ispravan program:
  - ne smije se oslanjati na sinkronizaciju prilikom globalne komunikacije,
  - mora pretpostavljati da globalna komunikacija *može* biti sinkonizirajuća.
- **Primjer 1** (izvodi se na dva procesa, 0 i 1):

```
switch(rank)
{   case 0:
        MPI_Bcast(buf1, count, type, 0, comm);
        MPI_Bcast(buf2, count, type, 1, comm);
    break;
    case 1:
        MPI_Bcast(buf2, count, type, 1, comm);
        MPI_Bcast(buf1, count, type, 0, comm);
    break;
}
```



- primjer je neispravan jer ukoliko je operacija sinkronizirajuća, nastaje potpuni zastoj
- rješenje: globalni pozivi moraju imati jednaki redoslijed za sve procese u grupi
- Primjer 2:

```
switch(rank)
{   case 0:
        MPI_Bcast(buf1, count, type, 0, comm);
        MPI_Send(buf2, count, type, 1, tag, comm);
    break;
    case 1:
        MPI_Recv(buf2, count, type, 0, tag, comm, status);
        MPI_Bcast(buf1, count, type, 0, comm);
    break;
}
```



- primjer je neispravan jer dolazi do potpunog zastoja ako proces 0 čeka na Bcast poziv procesa 1
- rješenje: relativni poredak globalnih i *point-to-point* komunikacija mora biti takav da ne smije doći do potpunog zastoja u i slučaju kada su obje vrste operacija sinkronizirajuće
- **Primjer 3** (izvodi se na tri procesa, 0, 1 i 2):

```
switch(rank)
{   case 0:
        MPI_Bcast(buf1, count, type, 0, comm);
        MPI_Send(buf2, count, type, 1, tag, comm);
    break;
    case 1:
        MPI_Recv(buf2, count, type, MPI_ANY_SOURCE, tag, comm, status);
```