13.09.2006.

Paralelne algoritme možemo da izvršavamo na paralelnim računarima.

Ako jedan veliki problem ne možemo da riješimo sami na jednom računaru, onda ga podijelimo na više manjih problema i podijelimo ih na više ljudi. Paralelni računari su se pojavili odmah poslije klasičnih (sekvencijalnih) računara.

U okviru samog procesora se vrši više paralelnih koraka. Dok jedna instrukcija započinje, druga se već priprema.

Sama paralelizacija se prepušta kompajleru. Računari koji rade sa više baza podataka imaju više procesora 4, 8, 12 i 16.

U SQL-u ima jedna instrukcija: select * from tabela, koja nije paralelizovana, ali ima pseudo komandu:

select /*+ parallel (t, 4) */ koja se nalazi pod komentarom
* from tabela t;

t → tabela, 4 → br. procesora

Ova komanda je paralelna. (iako je to pseudo komanda čim ima + kompajler zna da je to komanda za njega)

Šta bi trebali da definišemo za model izračunljivosti (na pr. Tjuringovu mašinu)? → elementarne transformacije suviše jednostavne

Atributi paralelnog modela izračunljivosti:

1° primitivne jedinice (osnovne aktivnosti računara – elementarne tipove podataka i instrukcije koje na njih utiču

2° mehanizam podataka (kako pamtiti, tj. pristupati podacima)

3° def. kontrolni mehanizam, odnosno mehanizam upravljanja (kako podijeliti problem na primitivne jedinice i kako izvršavati te instrukcije)

4° način komunikacije (ako imamo više uređaja nas interesuje kako oni razmjenjuju podatke i način na koji oni to rade)

5° mehanizam sinhronizacije (da pravi podaci dođu u pravo vrijeme)

Kod savremenih računara nam nije potreban korak 4° i 5°, dok kod paralelnih moramo koristiti 1) – 5). Savremeni računar radi po principima Fon Nojmana.

1° princip (koncept): Podaci i programi se čuvaju u memoriji.

Ovo ne važi kod Tjuringove mašine, jer se kod nje to čuvalo na traci.

2° princip: Instrukcija specificira same operande, (instrukcija nosi informaciju o operandima, npr. add a_1, a_2).

3° princip: Instrukcije se izvršavaju sekvencijalno, tj. instrukcije se izvršavaju jedna po jedna, redom kao što su zapisane, osim ako naiđemo na instrukciju grananja ili instrukciju skoka, tada se mijenja tok izvršavanja. Uopšte ne važi kod Tjuringove mašine.

4° princip: Reprezentacija instrukcija i podataka se ne razlikuje, tj. jednako zapisujemo i instrukcije i podatke. (Instrukcija može da služi kao podatak.)

Postoje različiti modeli paralelnih računara koji drastično odstupaju od principa savremenih računara. Ovi koncepti su bitni da bi uprostili sam rad, samo pisanje programa (znamo logiku) nezavisno od samih računara. Savremeni računar se sastoji od: CPU, memorije, ulazno-izlaznih uređaja i od jedne ili više magistrala koje povezuju ove 3 komponente.

Definisali smo skup instrukcija koje CPU može da i vrši (1°), a CPU ima kontrolnu jedinicu koja upravlja radom sistema (3°). Prvi njen korak je da "dohvati" instrukciju, a drugi da izvrši tu instrukciju. Ako imamo operande, imamo više koraka (dohvatanje jednog pa drugog operanda, same podatke pamtimo u memoriji (2°))

Arhitektura paralelnih računara

Postoji više klasifikacija paralelnih računara, ali mi ćemo obratiti pažnju na klasifikaciju koju je dao Flynn. On je izvršio podjelu prema broju tokova instrukcija, odnosno broju tokova podataka.

Tok instrukcija je niz instrukcija koje računar treba da izvrši.

Tok podataka je niz podataka na kojima se ove instrukcije izvršavaju.

Prema ovoj klasifikaciji imamo 4 modela:

1) SISD (single instruction stream single data stream)
2) MISD (multiple —"— —"—)
3) SIMD
4) MIMD

} modeli paralelnog računara

Model SISD (1 tok instrukcija, 1 tok podataka) se koristi kod savremenih računara.

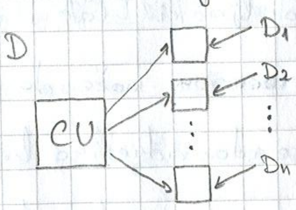
Kod modela 2) imamo na istom toku podataka više različitih instrukcija (više procesora za obradu podataka, odnosno n procesora (n>1)). Na pr. 3 procesora radi isti posao; nad istim podatkom provjeravamo da li je podatak X djeljiv sa 105=3·5·7. Imamo 3 procesora, pri čemu jedan procesor provjerava da li je X djeljiv sa 3, drugi sa 5, treći sa 7.

– protočna obrada

Možemo da angažujemo više procesora na 1 radnoj liniji.

3) i 4) model srijeću se najčešće u praksi.

– SIMD



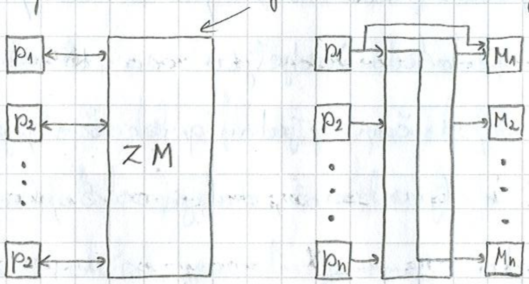
Kod 3) i 4) se javlja potreba za međusobnu razmjenu podataka i postoje 2 načina razmjene podataka:

1) preko zajedničkog adresnog prostora
– specijalni slučaj 1) je zajednička memorija

2) mrežni model – računari komuniciraju putem poruka preko mreže

Kod 1) načina računari imaju isti adresni prostor, odnosno računar može da pristupa jednoj adresnoj lokaciji.

Više procesora zajednička memorija.



Procesor p_1 može adresirati memoriju M_1, bitno je da imaju isti adresni prostor.

Postoje varijante u istom zajedničkom adresnom prostoru, gdje imamo više procesora, možemo imati i više memorija i imamo neki upravljački blok ili neki kanal preko koga za svaki procesor može da pristupi bilo kojoj memoriji (tj. može da adresira bilo koju lokaciju). Može i M_1 da bude za p_1, ali isto p_n može da pristupa M_1.

Imamo različite varijante pristupa i svaki procesor može da pristupi istoj lokaciji za isto vrijeme drugo različito vrijeme.

Pristup memoriji je mnogo sporiji od pristupa registrima u procesoru. Kod sekvencijalnih računara problem je brzina rada memorije.

Da bi se ubrzao rad, prave se manje memorije (brže), tkz. keš memorije (čim je manja memorija brže se pristupa toj memoriji).

Da li procesori mogu istovremeno da pristupaju istoj lokaciji?

U zavisnosti da li je ovo dozvoljeno ili ne, imamo R (mogućnost čitanja) i W (mogućnost upisa).

I model EREW (ni čitanje, ni upis istovremeno nije dozvoljeno).

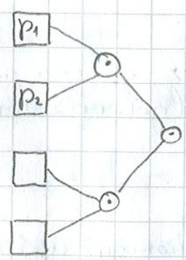
II model CREW (istovremeno možemo da čitamo, ali istovremeno ne možemo da upisujemo istu memorijsku lokaciju). Rijetko se srijeće.

III model ERCW najmanje interesantan (gotovo se ne srijeće).

IV model CRCW (istovremeno čitanje i istovremeni upis).

Možemo napraviti ograničenja kad imamo istovremeno upis.

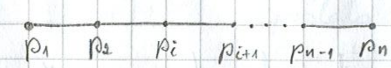
1. Dozvoljen upis ako svi upisuju isti sadržaj (tj. podatak).
2. Varijanta je da se upisuje vrijednost procesora koji ima najveći prioritet (prioritetni upis). Npr. procesor ima prioritet ako ima najmanji indeks.
3. Varijanta je da se upiše sadržaj koji je neka f-ja od svih mogućih argumenata koji treba da se upiše (tj. da ta f-ja bude simetrična u odnosu na svoje originala). Na pr. Σ, max (najmanja vrijednost od svih argumenata koje imamo).
Redosled argumenata nije bitan (x_1, x_2, x_3) (x_3, x_1, x_2)
4. Mogućnost da dozvolimo slučajan (proizvoljan) upis.
Moramo imati neki kontrolni mehanizam da se provjeri da li je to ta vrijednost.



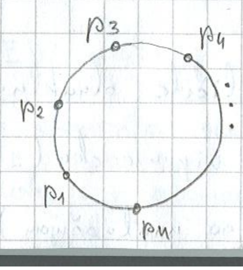
Na pr. za prvu mogućnost provjerava da li je vrijednost tih procesora ista.
Ako na jednom koraku ne dobijemo istu vrijednost, upis se prekida i javlja se greška.

Mrežni model

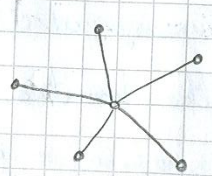
Procesore povezujemo preko komunikacionih kanala i oni komuniciraju slanjem poruka. Operacije su SEND i RECEIVE.
Jedan procesor šalje podatke koristeći komandu SEND drugom procesoru.
Ako imamo više procesora, moramo znati kom procesoru šaljemo ili ako primamo poruku, moramo znati od kog procesora je primamo (RECEIVE).
Potpuno povezan model - ako imamo n procesora i svaki je povezan sa svakim.
Mrežni model je:
1) linearan niz - (prvi i poslednji su povezani sa 1 procesorom, ostali sa 2)



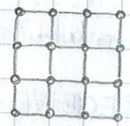
Može komunicirati sa onima sa kojima je povezan, inače šalje preko ostalih do odredišta.
2) ako p_1 i p_n žele da komuniciraju, onda ih možemo povezati u obliku prstena.



3) zvijezde



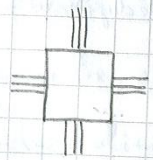
4) dvodimenzionalnog niza (matrice)



5) stabla itd.

Sve zavisi od toga kakve mogućnosti imamo i šta nam treba.

Transpjuter

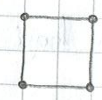


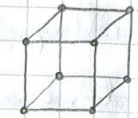
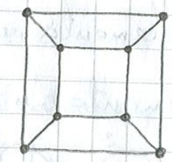
- nisu bili fizički povezani, ako ih jednom povežemo, ne mogu drugačije biti povezani.

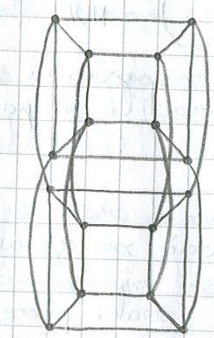
- pogodnost (nisu unaprijed povezani)

Prije startovanja programa može se definisati koji računar je sa kojim povezan.

Za različite probleme pravljenje različite strukture, na pr. za 16 procesora povezani u hiperkocku (2 kocke, spoljašnja sa spoljašnjom, unutrašnja sa unutrašnjom).

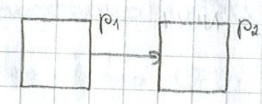
4 – 

8 –  

16 – 

Sinhronizacija između ovih procesora se ostvaruje koristeći SEND i RECEIVE.

Zato treba praviti takav algoritam da jedan procesor ne čeka dugo na poruku, tj. da vrijeme izvršavanja bude približno jednako.

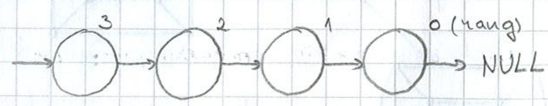


Na pr. ako procesor p_2 ne dobije poruku, a poslata mu je, onda on mora čekati poruku, sve dok je ne dobije.

Pretpostavimo da imamo jedan model sa zajedničkom memorijom.

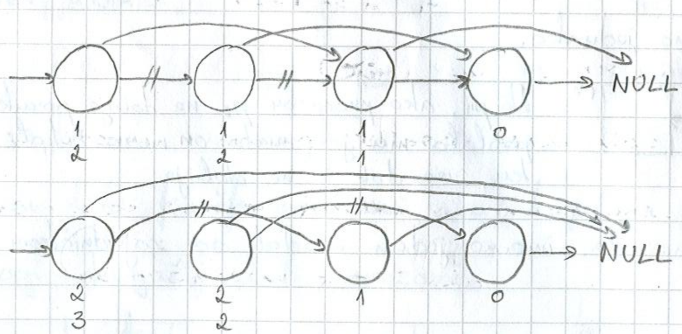
Imamo RAM model za sekvencijalni algoritam, ako dodamo mogućnost paralelizacije imamo PRAM i neka je još EREW. Data je ulančana lista, treba odrediti rang svakog elementa u toj listi. Možemo uzeti i neke pretpostavke: n elemenata, n procesora (1 procesoru 1 element).

Rang elementa u listi je broj elemenata koji su posle tog elementa u listi.



Za koliko ovo možemo uraditi na paralelnom računaru, odnosno na PRAM-u?

- Uvedimo dodatni pokazivač za rješavanje problema.
- Udvostručavamo korake, svaki procesor gleda iza sebe i svima stavlja 1, osim posledujeg koji je 0, jer gleda na NULL. Pa svaki sabira sa svojim sledbenikom.



Završava se kad svi procesori pokazuju na NULL.

21.09.2006.

Ono što nas interesuje kod paralelnog računara je vrijeme izvršavanja (od momenta kad je startovan prvi procesor do momenta kad je završio posledнji procesor).

Broj tih koraka nazivamo vremenskom složenošću tog algoritma.

Kod paralelnog računara moramo prebrojati:

- korake izračunavanja ili računske korake
- korake usmjeravanja ili razmjene podataka

Koraci usmjeravanje - vrijeme koje protekne da podatak od 1 procesora stigne na odredišno mjesto (do drugog procesora (CPU) ili memorije).

Ako jedna CPU šalje podatke drugom procesoru, ako nemamo direktnu vezu, mi podatak moramo da usmjerimo u pravom smjeru ili ako imamo zajedničku memoriju, prvi šalje u memoriju, a drugi uzima iz memorije.

Računski koraci izvršavaju se u jednom procesoru.

$\sigma(f(n))$ - kad želimo da damo gornju ocjenu

$\Omega(f(n))$ - —||— —||— donju ocjenu

$\Theta(f(n))$ - —||— —||— tačnu ocjenu

ubrzanje = $\frac{\text{vrijeme izvršavanja najbržeg poznatog sekv. algoritma}}{\text{vrijeme izvršavanja paralelnog algoritma}}$

$speedup = \dfrac{VINPSA}{VIPA}$

VINPSA < VIPA, pa je ubrzanje ≤ 1, tj. $S \geq 1$, $1 \leq S \leq p$

p broj procesora

Ako je ubrzanje $> p$, možemo konstruisati brži sekvencijalni algoritam, tako što bi simulirali rad paralelnog računara na jednom računaru.

$\boxed{p_1}\ \boxed{p_2}\ \boxed{p_3} \rightarrow$

Cijena paralelnog algoritma je vrijeme izvršavanja paralelnog algoritma puta broj procesora.

$$COST = T \times p$$

Produktivnost (efikasnost) rada je vrijeme izvršavanja najbržeg poznatog sekvencijalnog algoritma kroz vrijeme izvršavanja paralelnog algoritma.

$$e = efficiency = \frac{VINPSA}{COST}$$

$$0 < e \leq 1$$

Ako je $e = 1$ ubrzanje je 5 (koristili smo 5 CPU i 5 puta smo povećali brzinu). Cilj je da efikasnost bude što je moguće bliža.

Pretpostavimo da imamo PRAM, tj. paralelni računar sa direktnim pristupom. Problem koji rješavamo je:

10 redni broj u listi

Data je lista od n elemenata čiji su elementi smješteni u zajedničkoj memoriji.

next[i] - sledbenik od i

i-ti element je dodijeljen procesoru p_i $(i \rightarrow p_i)$

$$\rightarrow \underset{p_3}{\overset{3}{5}} \rightarrow \underset{p_1}{\overset{1}{7}} \rightarrow \underset{p_2}{\overset{2}{9}} \rightarrow nil$$

Nas interesuje da za svaki elemen izračunamo redni broj elementa (broj elemenata koji se nalazi u listi posle njega).

$$d[i] = \begin{cases} 0, & next[i] = nil \\ d[next[i]]+1, & inače \end{cases}$$

$o(n)$ - vrijeme koje zahtjeva ovaj algoritam.

Drugo rješenje je da dobijemo algoritam sa složenošću $o(\log n)$

LIST_RB( )

1° for $\forall$ procesor i

2° do if next[i] = nil

3° then $d[i] \leftarrow 0$

4° else $d[i] \leftarrow 1$

5° while ∃ procesor i takav da je $next[i] \neq nil$

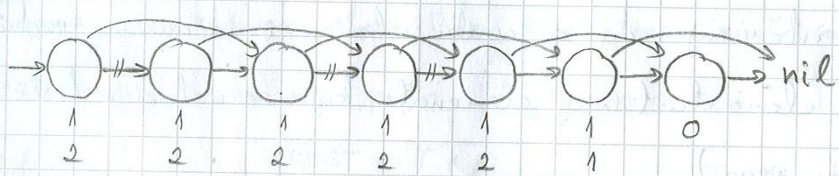
6° do for ∀ procesor i

7° do if $next[i] \neq nil$

8° then $d[i] \leftarrow d[i] + d[next[i]]$

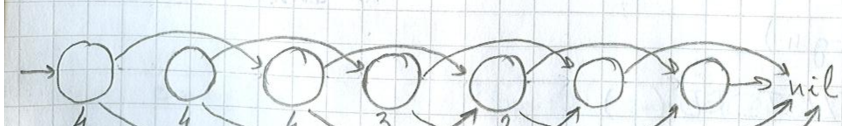
9° $next[i] \leftarrow next[next[i]]$

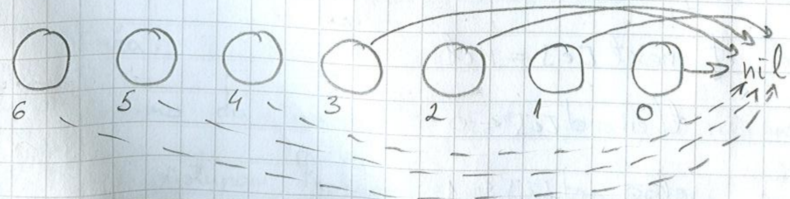
P4. Imamo PRAM, EREW (samo jedan procesor u principu pristupa samo jednoj lokaciji).



8° $\begin{cases} t \leftarrow d[next[i]] \\ d[i] \leftarrow d[i] + t \end{cases}$

9° $\begin{cases} n \leftarrow next[next[i]] \\ next[i] \leftarrow n \end{cases}$



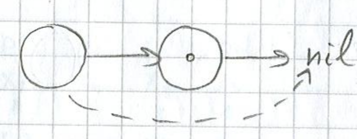


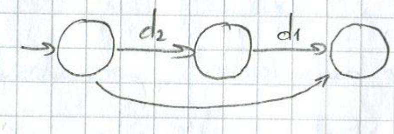
Za petlju važi sledeća invarijanta: $next[i] = nil$

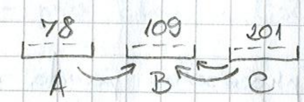
$\wedge$ $d[i]$ rastojanje do kraja (ima pravu vrijednost)

$\wedge$ ($next[i] \neq nil$ i ukazuje na objekat koji je na rastojanju $d[i]$)

- Ova varijanta važi za S-ti prolaz, a treba dokazati da važi i za S+1 prolaz:







I iz A prebacujemo 1, a iz C dvije

II B u A dvije, a iz B u C jednu

III iz A uzmemo proizvoljan broj kuglica i prebacimo u C

IV 4 kuglice iz C prebacujemo u A

V iz A izbacujemo po 5 kuglica

Da li ovakvim izbacivanjem možemo B i C da ostavimo prazne?

Ovaj zadatak ne možemo riješiti.

Invarijanta zadatka: Koje god pravilo da koristimo $B:3$ daje ostatak 1.

Ako je $i \neq j$ onda je $next[i] \neq next[j]$ ili je $next[i] = next[j] = nil$

$T_p(n) = \sigma(\log n)$ – složenost

cijena $= \sigma(n \log n)$

$T_s(n) = \sigma(n)$

ubrzanje $S = \frac{n}{\log n}$

$E = \frac{n}{n \log n} = \frac{1}{\log n} \xrightarrow[n\to\infty]{} 0$

Možemo li smanjiti broj procesora pa da efikasnost bude veća?

Broj CPU smanjimo na $p = \frac{n}{\log n}$ i svakom CPU dodijelimo $\log n$ čvorova.

28.09.2006.

Neka je $*$ asocijativna binarna operacija. Na osnovu niza $x_1, x_2, \dots, x_n$ izračunati niz $y_1, y_2, \dots, y_n$ po formulama:

$y_1 = x_1$

$y_k = y_{k-1} * x_k = x_1 * x_2 * \dots * x_k$, $k = 2, 3, \dots, n$

Imamo n procesora i svaki procesor sadrži jedan član ovoga niza X. Procesor p_i sadrži član $X[i] = x_k$.

p_1	p_2	p_3	p_4
x_2	x_4	x_1	x_3
$\parallel$	$\parallel$	$\parallel$	$\parallel$
$X[1]$	$X[2]$	$X[3]$	$X[4]$

Pretpostavka, članovi niza su povezani u listi.

x_1 – prvi član u listi

x_2 – drugi član u listi

Kako napraviti paralelni algoritam?

Ako po $x_i = 1$ i za $*$ uzmemo oper. $+$, računanje prefiksa svodi se na računanje $d[i]$. Uvodimo oznaku:

$$[i,j] = x_i * x_{i+1} * \dots * x_j$$

$[k,k] = x_k$

$[i,k] = [i,j] * [j+1,k]$, pri čemu važi $1 \leq u \leq j < k \leq n$

$y[i] = y_k = [1,k]$ – interval

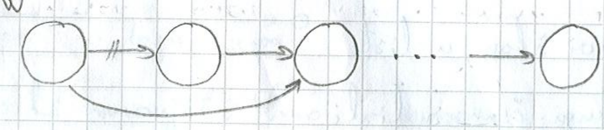
p_i sadrži $X[i] = x_k$ kad važi ovo gore

Algoritam: ideja - povećavamo pokazivač za istu dužinu

list_prefiks(x)

```
1° for ∀ procesor i
2° do y[i] ← x[i]
3° while (∃ i) next[i] ≠ nil
4°       do for (∀ procesor i)
5°             do if next[i] ≠ nil
6°                   then y[next[i]] ← y[i] * y[next[i]]
7°                   next[i] ← next[next[i]]
```

EREW

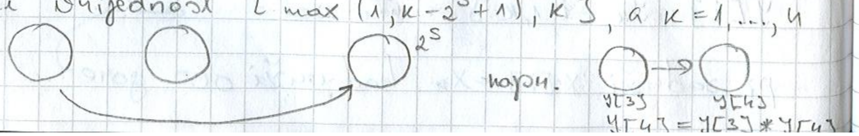


Potrebna sinhronizacija, možemo da realizujemo na mašini EREW.

Složenost $O(\log n)$

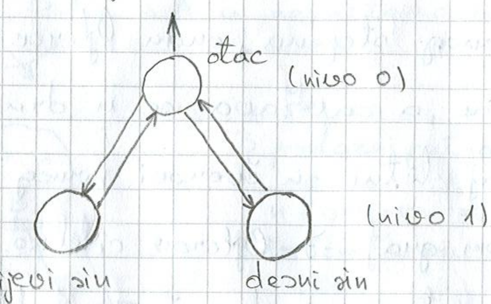
Složenost ⇒ iz invarijante

Posle s-tog ponavljanja ciklusa pokazivači obuhvataju 2^s čvorova liste ili su nil, a k-ti element liste pamti vrijednost $[\max(1, k-2^s+1), k]$, a $k = 1, \dots, n$



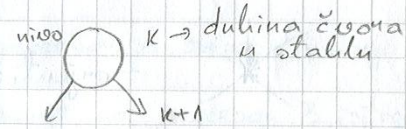
npr.

Ojlerov ciklus

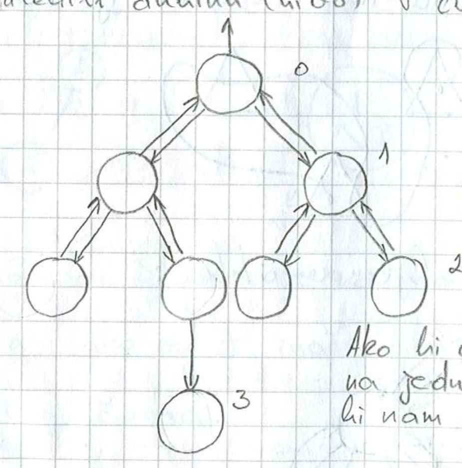


(parent[i], left[i], right[i])

(p[i], l[i], r[i])



Pr.: Odrediti dubinu (nivo) ∀ čvor u datom stablu.



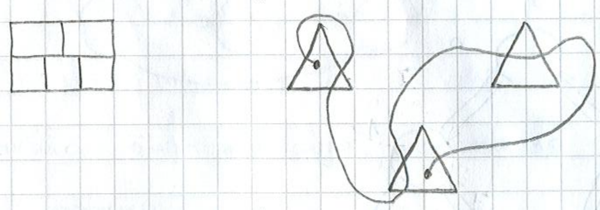
Ako bi ovaj zadatak rešavali na jednom računaru, trebalo bi nam $O(n)$ vremena.

Ojlerov put u grafu je put koji svakom granom grafa prolazi tačno jednom, a može proći više puta kroz čvor. Ojlerov put koji počinje i završava se u istom čvoru naziva se Ojlerov ciklus. Ako imamo 2 čvora neparnog stepena, onda Ojlerov put počinje u jednom čvoru, a završava se u drugom. Za neorjentisan graf bitni su čvorovi parnog stepena.

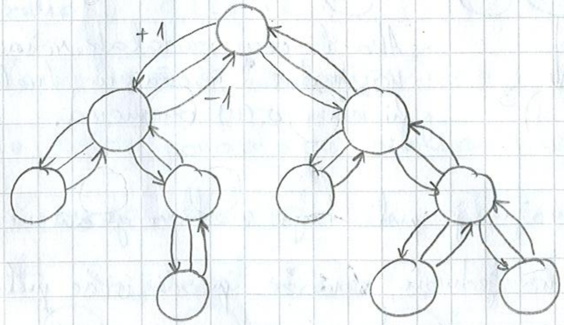
T: U orjentisanom graf ∃ Ojlerov ciklus akko ulazni i izlazni stepen svakog čvora poklapa. (graf mora biti i povezan)

– orjentisan graf

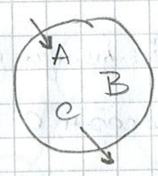
Svaka ulazna grana ima izlaznu granu.



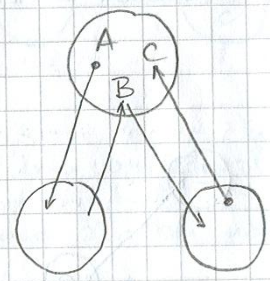
Treba konstruisati Ojlerov ciklus:



Svakom čvoru ćemo dodijeliti 3 procesora A, B i C.



Kada dolazimo u čvor, uvijek dolazimo u A a napuštamo u C.

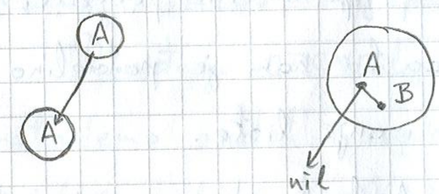


B prolazni put od A do C.

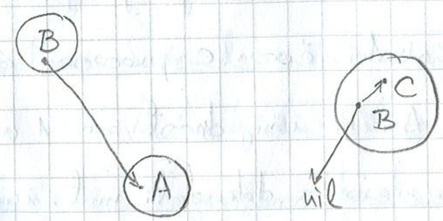
n čvorova ⇒ $3n$ procesora

i-tom čvoru odgovaraju $3i, 3i+1, 3i+2$

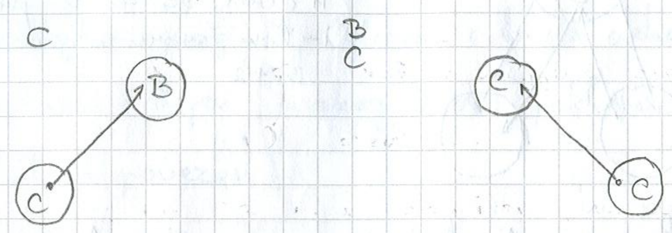
I) Procesor A ukazuje na procesor A svog lijevog sin, ako on ∃, inače ukazuje na procesor B istog čvora.



II) Procesor B ukazuje na procesor A svog desnog sina, ako on ∃, inače ukazuje na procesor C istog čvora.



III) Procesor C ukazuje na procesor B svoga roditelja, ako je on lijevi sin, a na procesor C svoga roditelja ako je on desni sin. Procesor C korjena ukazuje na nil.

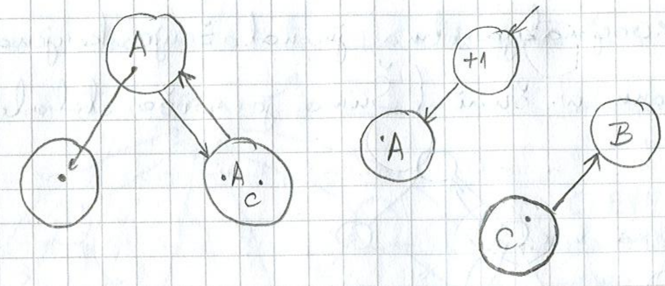


Za koliko vremena možemo formirati Ojlerov ciklus?

Možemo svaki procesor paralelno povezati sa svojim sledbenikom, a za to nam je potrebno $O(1)$ vremena. → za formiranje liste.

Kad ulazi procesor A treba da ima +1 vrijednost, a C −1 kad izlazi.

Ako procesoru A dodijelimo +1, B je 0 i C = −1 i primjenimo algoritam za računanje prefiksa $* = +$ i dobićemo da za svaki čvor C procesor će sadržati dubinu čvora, a A i B vrijednost za 1 veću od samog nivoa. Ovo možemo dokazati mat. indukcijom.



C ima vrijednost kao i B kad je u lijevom sinu, samo što su na različitim nivoima.

Složenost $O(\log 3n) = O(\log n)$

Ovo radimo na EREW računaru:

efikasnost $E = \frac{n}{3n \cdot \log n} = \frac{1}{3 \log n} \Rightarrow$ slaba efikasnost, veliki broj procesora.

Poređenje CRCW algoritma sa EREW algoritmom

Razmotrićemo model sa upisom ist vrijednosti.

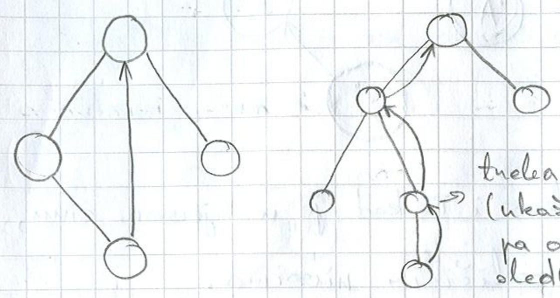
CRCW brže i lakše realizujemo od EREW (bolji za praktičnu realizaciju).

Između procesora i memorije moraju biti logička kola, i mogu biti veoma složena. U praksi skoro ne ∄ ovakav model, postoje njegove aproksimacije.

Moć konkurentnog čitanja − CR

Ilustrovaćemo algoritam konkurentnog čitanja (CR).

Primjer ovakvog algoritma je nalaženje korijena svakom čvoru u šumi. (Šuma je više stabala)



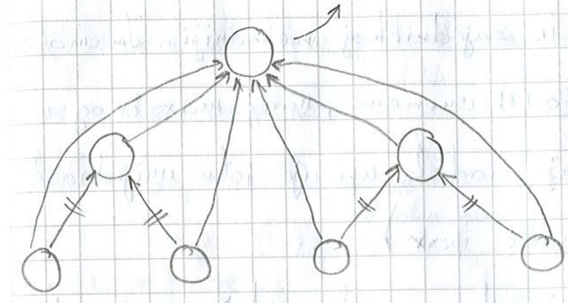
treba naći njegov korijen? (ukažimo na njegov sledbenik pa onda na sledbenik njegovog sledbenika)

Svaki čvor treba da ima pokazivač na svoj korijen. Stablo zadajemo tako što zadajemo da svaki čvor ima pokazivač na oca. (root - korijen)

Algoritam → za konkurentno čitanje CRW
find_roots (F) (naći korijen u šumi F)

1° for ∀ procesor i
2° do if $p[i] = nil$
3° then $root[i] \leftarrow i$
4° while (∃ procesor i) $p[i] \neq nil$
5° do for (∀ procesor i)
6° do if $p[i] \neq nil$ (ako je nil već smo ga odredili)
7° then $root[i] \leftarrow root[p[i]]$
8° $p[i] \leftarrow p[p[i]]$

Ovdje nema ekskluzivnog čitanja, zato što će se desiti da dva procesora čitaju istu vrijednost.



Složenost $o(\log d)$ gdje je d - visina našeg stabla

CREW - se koristi
$d \geq o(\log n)$ - najbolja moguća visina

Koraci 7° i 8° zahtjevaju konkurentno čitanje. Ako imamo n čvorova, onda je $d \geq o(\log n)$. Složenost bi se ponašala kao $o(\log \log n)$. Ako bi ovo realizovali na bilo kom računaru koji ne zadovoljava konkurentno čitanje, složenost našeg algoritma je $\Omega(\log n) \Rightarrow$ manja složenost od log n.
donja ocjena

Na svakom nivou mi udvostručavamo broj čvorova.

Invarijanta je sledeća:

Posle s-tog prelaza kroz ciklus ili je $p[i] = nil$ i $root[i]$ je pokazivač korijena ili je $p[i]$ predak na rastojanju 2^s.

5.10.2006.

CW - nalaženje max u nizu

Imamo niz $A[1,\dots,n]$ u zajedničkoj memoriji. Imamo n^2 procesora. Treba za $\sigma(1)$ vremena, naći max ovog niza na mašini CRCW, ovaj model upisuje istu vrijednost (u jednom koraku naći max)

Svaki procesor P_{ij} može da uporedi sadržaj u nizu $A[i]$ i $A[j]$. Ako je $A[i] < A[j]$, onda $A[i]$ nije max. Onda imamo još jedan dodati niz $m[i]$ i on sadrži T ili ⊥.

Ako je $m[i] = true$, onda je $A[i] = max$ element i možemo napisati sledeći algoritam:

fast_max(A)

1° $n \leftarrow$ length(A) - zadavanje dužine niza A

2° for $i \leftarrow 0$ to $(n-1)$ (paralelno)

3° do $m[i] \leftarrow$ true ⌜u početku su svi kandidati za max element⌟

4° for $i \leftarrow 0$ to $n-1$ and $j \leftarrow 0$ to $n-1$ (paralelno)

5° do if $A[i] < A[j]$

6° then $m[i] \leftarrow$ false → korak za istovremeni upis

7° for $i \leftarrow 0$ to $n-1$ (paralelno)

8° do if $m[i] =$ true

9° then max ← $A[i]$

10° return max

U 5-om koraku je konkurentno čitanje t_j. P_{ij} i P_{ji} traže iste vrijednosti.

Ako napr. $P_{3,7}$ i $P_{3,9}$ oba ova procesora će pokušati da u $m[i] =$ false

Da bi našli max, moramo uporediti sve elemente, potrebno je $\sigma(n)$ → složenost na sekvencijalnom algoritmu.

Složenost $T(n) = \sigma(1)$ $\quad T_s(n) = \sigma(n)$

Cijena $C(n) = \sigma(n^2)$

Efikasnost $= \frac{n}{n^2} = \frac{1}{n} \to 0$ efikasnost je vrlo slaba.

Da li se može poboljšati ova efikasnost?

Mi imamo n^2 procesora. P_{ij} i P_{ji} $(1 < i < j \leq n)$ upoređuju iste članove. Uz malu modifikaciju, to sve može da radi isti procesor.

$\binom{n}{2}$ potrebni procesori

Treba modifikovati tako da se prvo uporede $a_i \neq a_j$ i vidi se koji je manji:

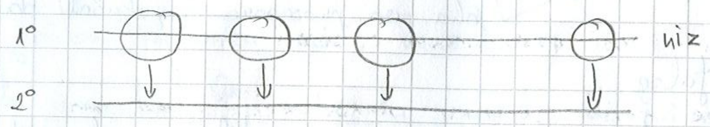
$$a_i < a_j \quad \xrightarrow{DA} m[i] = \perp \qquad \xrightarrow[NE]{} m[i] = \perp \qquad \binom{n}{2} = \frac{n(n-1)}{2} = \sigma(n^2)$$

1. Ako imamo $n^{3/2}$ procesora, naći max niza za $\sigma(1)$ vr-

emena na CRCW mašini.

1° čitav niz se podijeli na više manjih grupa i nađe se max niza

2° nalaženje max manjih članova



Podijelimo elemente niza u $\sqrt{n}$ grupa sa po $\sqrt{n}$ elemenata. Svakoj grupi pridružimo po n procesora, tj. $(\sqrt{n})^2$. Koristeći prethodno opisani algoritam, naći max svake grupe za $O(1)$ vremena. Takvih max ima $\sqrt{n}$, odnosno formirali smo niz dužine $\sqrt{n}$.
Iskoristimo n procesora da nađemo max ovog niza za $O(1)$ vremena. Dobijeni max je max polaznog niza.

- 1) $\sqrt{n}$ grupa, svakoj grupi pridružimo n procesora, dakle $\sqrt{n} \cdot n = n^{3/2}$
- U drugom koraku biće neiskorišćeno $n^{3/2} - n$.

Sada je efikasnost $T(n) = O(1)$ $E(n) = \frac{n}{n^{3/2}} = \frac{1}{\sqrt{n}}$

Domaći:

(5b) 1. Neka je $\varepsilon > 0$ fiksirani broj. Konstruisati algoritam koji će naći max niza dužine n koristeći $O(n^{1+\varepsilon})$ procesora za $O(1)$ vremena, koristeći CRCW.

(10b) 2. Ispisati precizno algoritam (pseudo kod) $O(1)$ vremena, naći max
p - broj procesora
n - broj elemenata u nizu
[ne može se više puta ići u dubinu]

Modeliranje CRCW računara pomoću EREW računara

- mašina CRCW je moćnija od EREW mašine.

T: $\forall$ CRCW algoritam (u modelu sa upisom iste vrijednosti) koji koristi p procesora, $\exists$ EREW algoritam za isti zadatak sa istim brojem procesora za koga vrijeme izvršavanja nije veće od $O(\log p)$ puta vremena izvršavanja polaznog algoritma.

t - vrijeme izvršavanja polaznog algoritma

$t \cdot \log p$ - vrijeme izvršavanja na EREW mašini

$$t \leq t \cdot \log p$$

Modeliramo CW, sam CR

l, x $\quad$ l - adresa (I polje), x - vrijednost (II polje)

1° Ako procesor p_i želi da upiše vrijednost x_i na lokaciju l_i, onda on niz na poziciju $A[i]$ upisuje par (l_i, x_i).

2° Sortiramo niz A_i po I koordinati l_i

3° Procesor p_i upoređuje adresu koju on zaduži l_i' sa adresom koju zaduži procesor p_{i-1} l_{i-1}. Ako je $l_i' = l_{i-1}'$ onda procesor p_i ništa ne radi, a ako je $l_i' \neq l_{i-1}'$, onda procesor p_i upisuje x_i' na lokaciju l_i'

Specijalno, procesor p_1 ništa ne radi, pošto nema šta da se upoređuje, on će na l_1' upisat x_1'.

na pr.

2	4
2	4
2	4
5	9
5	9
7	11
7	11
7	11

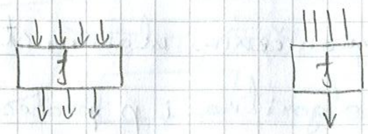
Na istim lokacijama upisuje se ista vrijednost.

Ovo je niz dužine 5 i možemo ga paralelno sortirati za $o(\log p)$ vremena na EREW mašini.

Teorema Brenta i povećanje efikasnosti

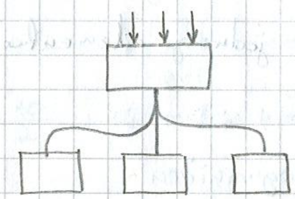
– funkcionalni elementi FE

– više ulaza i više izlaza



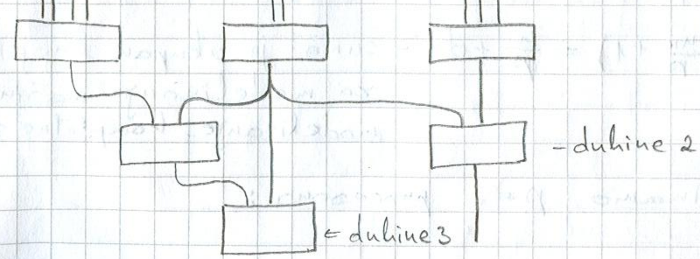
Međutim nas interesuje FE sa jednim izlazom i posmatraćemo shemu iz FE kod koje nemamo ciklus (SFE).

Broj elemenata kod kojih izlaz predstavlja ulaz naziva se izlazni stepen FE.



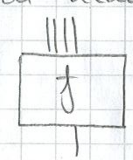
Veličina sheme je broj FE u shemi.

Dužina elementa u shemi je najduži put od ulaza do izlaza iz tog elementa, a dužina sheme je najveća dužina elementa u shemi.



Teorema (Brenta): Za modeliranje rada sheme dužine d i veličine n sa ograničenim ulaznim stepenom elementa pomoću CREW algoritma i p procesora dovoljno je $o\left(\frac{n}{p} + d\right)$ vremena. Pošto nema ciklusa, a na dužini smo k, mi se ne vraćamo više na dužinu k-1, a ne možemo preći na k, dok ne sračunamo na k-1.

Ako su neki elementi na dužini k, onda ih možemo obrađivati paralelno, jer izlaz jednog elementa ne zavisi od ulaza 2 elementa.



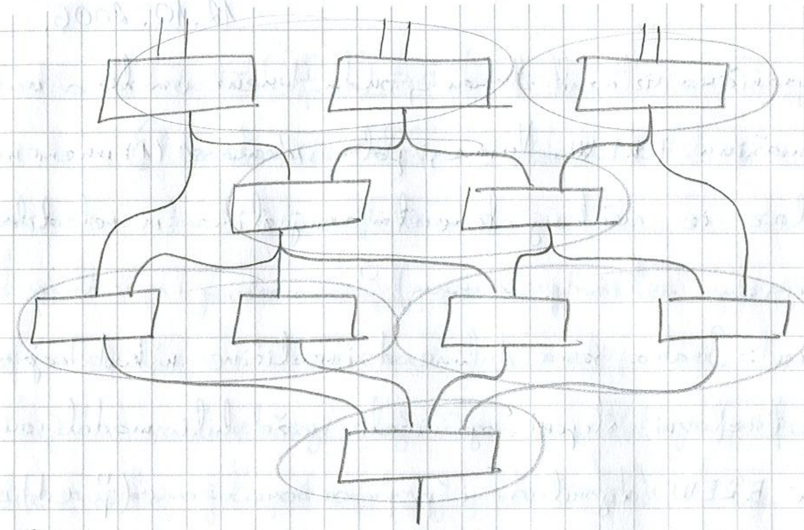
→ broj ulaza je ograničen

Ako na k-tom nivou imamo n_k FE onda za računanje vrijednost FE treba cio broj.

$$\left\lceil \frac{n_k}{p} \right\rceil \le \frac{n_k}{p} + 1$$, jer n_k nije cio broj

$$\sum_{k=1}^{d} \left(\frac{n_k}{p} + 1\right) = \frac{n}{p} + d$$ - suma je ukupan broj koraka za modeliranje mašine, tj. za modeliranje kompletne sheme

Ako imamo $p = 2$ procesora:



Potrebno je 6 koraka

$d = 4$ (nivoi) $\quad n = 10 \quad p = 2$

$$\frac{10}{2} + 4 = 5 + 4 = 9$$ - ovo je najmanje koraka