12.10.2006.

Ako ograničimo izlazni stepen, onda Brentova lema važi i za mašinu EREW. Naime, potrošićemo $O(1)$ vremena da izlaz iz svakog elementa smjestimo u nekoliko lokacija.

Posledica: Svaka šema dužine d i veličine n kod koje su ulazni i izlazni stepeni ograničeni može biti modelirana pomoću EREW algoritma i p procesora za $O(\frac{n}{p}+d)$ vremena.

T: Ako algoritam A koristi p procesora i ima vremensku složenost t, tada $\forall p' < p$ $\exists$ algoritam A' za isti zadatak koji koristi p' procesora i ima vremensku složenost $O(\frac{pt}{p'})$.

D: Korake možemo numerisati sa $1, 2, \dots, d$ (u svakom koraku koristimo najviše t procesora). Algoritam A' modelira svaki korak algoritma A za $O(\lceil\frac{p}{p'}\rceil)$ vremena kao i Brentovoj lemi. Koristimo p procesora

$k: p_1, p_2, \dots p_p$

gledamo k-ti korak. Uzmimo $\lceil\frac{p}{p'}\rceil$ jer možda ne iskoristimo sve procesore.

Ukupno vrijeme izvršavanja algoritma A' je:

$$t \cdot O(\lceil\frac{p}{p'}\rceil) = O(\frac{t \cdot p}{p'})$$

Efikasno paralelno računanje prefiksa

Imamo n procesora $(p=n)$ i problem smo rešavali za $O(\log n)$ vremena.

Cijena paralelnog algoritma je $O(n\log n) \to$ (vrijeme izvršavanja puta broj procesora)

Sekvencijalni algoritam radi sa $O(n)$ vremena.

Efikasnost: $E(n) = \frac{n}{n\log n} = \frac{1}{\log n} \to 0$ | mala efikasnost zbog velikog broja procesora, a da bi se povećala smanjiti broj procesora)

Mi da iskoristimo manji broj procesora

$$p = \Theta(\frac{n}{\log n})$$

Svaki procesor umjesto da obradi 1 element, obrađuje $\frac{n}{p} = \Theta(\log n)$ elemenata.

$\exists$ algoritam koji očigledno radi efikasno i bazira se na vjerovatnoći.

Ideja: Isključiti neke elemente, obradimo rekurzivno ostale elemente, pa se vratimo na obradu isključenih elemenata. Mi ćemo u x_n smjestiti $x_n * x_{n+1}$ prije isključenja i smještamo vrijednost u njegovog sledbenika.

Pravila prilikom isključenja elemenata:

1) Ne smijemo istovremeno isključiti 2 elementa koji odgovaraju istom procesoru (da ne bi isključili sve elemente iz procesora, pa onda on ne bi imao šta da radi.)

2) Ne smijemo isključiti 2 susjedna elementa iz liste (jer susjedni elementi pamte vrijednost isključenog elementa)

Odabir elemenata koje treba isključiti uraditi brzo za $o(1)$ vremena i da su istovremeno zadovoljeni 1) i 2).

Postoji i jedan vjerovatni algoritam ovakvog isključivanja elemenata i 3 koraka: (Bazira se na zakonima vjerovatnoće)

1) Svaki procesor slučajno bira jedan od svojih elemenata.

2) Svaki procesor baca novčić i sa vjerovatnoćom 1/2 označava izabrani element za izbacivanje.

3) Označeni k-ti element se udaljava ako next[k] nije označen.

* Izračunati, tj dokazati da je očekivano vrijeme sravnjivanja $o(\log n)$ vremena.

Napomena: Ako imamo da svaki procesor ima više elemenata, koja je vjerovatnoća da jedan izabrani element bude izbačen. Koja je vjerovatnoća da je njegov sledbenik označen za izbacivanje (manja 1/2, sa vjerovatnoćom > 1/4 je da će ovaj element biti udaljen).

X_{k-1} X_k X_{k+1} - izbacimo X_k, a na to mjesto stavljamo $X_k * X_{k+1}$

Poziva se algoritam za izbacivanje prefiksa,

X_k $\square$ $X_k * X_{k+1}$, kad se završi rekurzija biće

y_{k-1} $\overset{y_k}{\square}$ y_{k+1}

→ prava vrijednost

Izračunati y_k. $y_k = y_{k-1} * X_k$, jer je u njegovom prethodniku sračunata vrijednost.

(Ovo je samo jedan korak, jer je X_k već sačuvan. Ako smo izbacili X_k, njegove susjede ne možemo izbaciti.)

$$E(n) = \frac{n}{\frac{n}{\log n} \log n} = 1$$ efikasnost računanja prefiksa

Bojenje grafa i max nezavisni skup

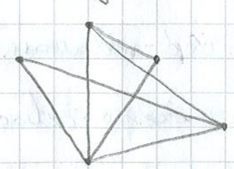
Ako 2 procesora pristupaju istom objektu (memorijskoj lokaciji) da čitaju, a CR nije dozvoljeno, pitanje je kom procesoru dozvoliti da prvi pročita ovaj element?

Jedan od načina je vjerovatnosni: Svaki od procesora "baca novčić" i u zavisnosti šta "padne" on pristupa ili ne pristupa. (U slučaju konflikta opet se baca novčić sve dok jedan ne dobije prednost.) Možemo dati deterministički algoritam za pristup elementu.

– procesor gleda index i onaj sa manjim indexom ima prednost, više procesora se bore za isti resurs.

Ilustrovaćemo pr.: Bojićemo listu sa 6 boja i potom pravimo nezavisan skup.

Def: Bojenje neorjentisanog grafa $G=(V,E)$ je preslikavanje $c: V \to N$ za koje važi $(\forall(u,v)\in E)\, c(u) \neq c(v)$



Mi treba kad imamo graf da, obojimo vrhove, tako da 2 susjedna vrha imaju $\neq$ boju.

Ako je kardinalnost $|C(V)| = k$ onda kažemo da smo graf obojili sa k-boja. Ponekad je pogodno slikati u skupu

$N_0 = N \cup \{0\}$.

* Koristeći paralelni algoritam obojiti graf u obliku liste.

Interesuje nas što je moguće manje boja. (listu obojimo sa najmanje 2 boje)

Primjetimo da se ova lista oboji sa 6 boja, skoro za konstantno vrijeme.

Konstruišemo za bojenje liste niz bojenja $C_0, \ldots, C_m$, pri čemu C_{k+1} konstruišemo pomoću C_k i C_{k+1} koristi manje boja od C_k.

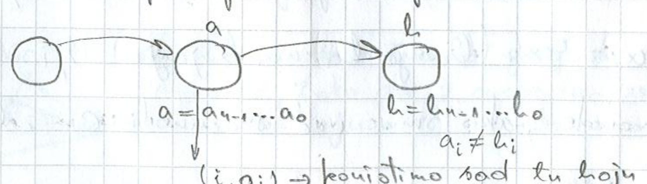
Pretpostavimo da imamo n procesora i n elemenata i svaki procesor zna svoj index. Ako neki procesor zaduži element x, onda njegov index označavamo sa $P(x)$. C_0 koristeći n boja i to su indexi procesora, tj. $C_0(x) = P(x)$. (Ovo je lazno bojenje) Opisati induktivni korak, ako smo već sračunali C_k i treba izračunati C_{k+1}.

Pretpostavimo da u bojenju C_k boje možemo zapisati sa r-bita.

Na pr. Neka je $C_k(x)=a$, a $C_k(next(x))=b$, pri čemu je $a=(a_{n-1}a_{n-2}\dots a_1a_0)_2$, $b=(b_{n-1}b_{n-2}\dots b_1b_0)_2$ (u binarnom brojnom sistemu): a i b su boje i ovo je njihova binarna reprezentacija. Pošto su x i $next\,x$ različito obojeni, boje a i b u različite, jer su boje susednih elemenata $\Rightarrow$ da $(\exists i\in\{0,\dots n-1\})$ takav da je $a_i\neq b_i$.

Kao nova boja za x, odnosno $C_{k+1}(x)$ definisaćemo kao par $(i,a_i)_2$, $\lceil C_{k+1}(x)=(i,a_i)_2\rceil$, (ako ima više indexa možemo uzeti prvi ili bilo koji od njih), tj. gledamo binarnu reprezentaciju zapisa i dopišemo cifru a_i.

Bojenje za poslednji element u listi (nema sledbenika) je $(0,a_0)_2$, tj. to je boja od a_0, odnosno od bojenja C_k dobili C_{k+1} (biće ili 1 ili 0) (Ne mijenja se vremenski, svaki put je ista cifra).



Sa koliko bita možemo zapisati novu boju?

Pošto je $i\in\{0,1,\dots,n-1\}$ za binarno zapisivanje i nama treba $\lceil \log n\rceil$ bita, a pošto dopisujemo cifru a_0 treba nam još bit, pa nam je potrebno najviše $\lceil \log n\rceil+1$ bita za zapisivanje boja u C_{k+1}. (smanjeno u odnosu na C_k)

Ako je $n\geq 4 \rightarrow \lceil \log 4\rceil+1=2+1=3$

$n=3 \rightarrow \lceil \log 3\rceil+1=2+1=3$ (ne možemo više smanjiti broj bita)

Ako je $n=3$ sve je zapisano sa 3 bita $a_2a_1a_0$. Naši kodovi mogu da počinju sa $\underset{(\text{index})}{i}\in \frac{00}{0}\ \frac{01}{1}\ \frac{10}{2}$ (dopisujemo 0 ili 1 u zavisnosti od toga koja je naša cifra i imaćemo 6 boja (2×3)).

110, 111 } kad su 3 bita, ove 2 boje otpadaju, tj. ne dozvoljavamo 110 i 111. Koristimo od 0 do 5, to su naše boje. (000, 001, 010, 011, 100, 101).

Koliko nam treba koraka do finalne boje?

$c_0,\dots,c_m \quad m=?$

Formiranje jedne boje ide od $O(1) \Rightarrow$ za sva bojenja nam je potrebno $O(m)$ koraka, odnosno koliko je $\neq$ bojanja. Ako u i-tom koraku koristimo n_i bita, onda važi da je $i\leq \lceil \log n_{i-1}\rceil+1$ (n_2 manji od prethodnih koraka.

T: $u_i < \lceil \log^{(i)} n \rceil + 2$ za $\lceil \log^{(i)} n \rceil \geq 2$

(na i-tom koraku ovo važi)

D: matematičkom indukcijom

$\log^{(i)} n$ – višestrukost (ugnježdenost) ovog algoritma tj.

$$\log^{(i)} n = \underbrace{\log(\log \ldots (\log n))}_{i\text{-puta}}$$

Treba izabrati tako i da log bude < 2.

Def: $\log^* n$ je minimalno, tj. $\log^* n = \min\{ i \geq 0 : \log^{(i)} n \leq 1\}$

iz T $\Rightarrow$ $m = o(\log^* n)$ → složenost našeg algoritma

Ovo je skoro konstantno vrijeme, jer je $\log^* n$ mali broj

$n \leq 2^{65536} \Rightarrow \log^* n \leq 5$ koraka, to je onda konstantno vrijeme. Da bi imali 6 koraka mora biti

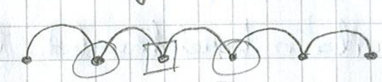
$$n \leq 2^{65536} = 4^{65536}$$

Def: Skup $V' \subseteq V$ čvorova grafa $G = (V, E)$ nazivamo nezavisnim skupom, onda $(\forall (u,v) \in V')$, $(u,v) \notin E$, tačnije nikoja 2 vrha iz V' nijesu spojena granama iz skupa E.

Nezavisni skup V' nazivamo max nezavisnim skupom (MNS) ako dodavanjem bilo kog vrha iz $V \setminus V'$ on prestaje da bude nezavisan skup. (ne treba ovo miješati sa najvećim nez. skupom)

Nalaženje najvećeg nez. skupa je polinominalne složenosti (NP)?

Nas interesuje max nezavisni skup.



Ako pogledamo 3 elementa (susjedna), onda bar 1 element pripada max nezavisnom skupu, tj. svaki max nezavisan skup sadrži više od $\frac{n}{3}$ elemenata. Prvo obojimo listu sa 6 boja i onda svaki element liste označimo sa tačno, tj. $n[i] = T$ (nismo ga još isključili, odnosno može ga uključiti u max nezavisan skup). $\forall$ od 6 boja ponovimo sledeće korake (to radimo paralelno):

– Ako razmotrimo boju k, onda gledamo da li je $C(x) = k$ i da li je $n(x) = T$. Ako su ispunjena ova 2 uslova, onda mi x dodajemo u MNS, a njegovog prethodnika i sledbenika, a $(n(next(x)) = \bot$ i $n(prex(x)) = \bot$ označimo da ih ne možemo uključiti u MNS i tako ponovimo $\forall$ 6 boja. Koraci se izvršavaju paralelno, odno svo procesori izvršava ove korake.

Koliko nam treba vremena?

Ukupno nam treba $O(1)$ vremena.

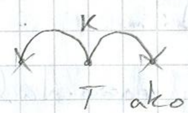
Složenost algoritma $O(\log^* n)$ - za početni korak, da bi listu obojili sa 6 boja.

Dokazati da je algoritam korektan (da radi), tj. da izdvaja max nez. skup. Svako x je obojeno nekom bojom i bar jedan od 6 koraka će doći na razmatranje.

Ako je T - uključen je, a ako je $\perp$ nije. jer je njegov susjed uključen.

Ako razmatram x obojen sa k i gledamo da je

$$h(x) = \begin{cases} T - \text{uključen je} \\ \perp - \text{samo ako je njegov susjed uključen} \end{cases}$$



ako je T susjedi su isključeni.
(Susjedi ne mogu biti uključeni!)

26.10.2006.

Data broadcasting

Ako jedan procesor želi da preda podatke ostalim procesorima.

Predaja podataka:

1) jedan-svima: kada jedan procesor želi da proslijedi svoj podatak svim ostalim procesorima.

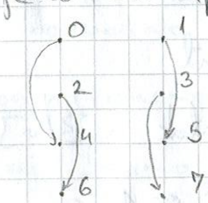
2) svaki-svakom: $\forall$ od p procesora šalje svoj podatak svim ostalim procesorima.

Ovo možemo da realizujemo na CREW i CRCW mašinama.

Složenost 1° je $O(1)$

2° je $O(p)$ vremena za CREW.

Ako ovo realizujemo na EREW mašini 1° slučaj realizujemo za $O(p)$ vremena, ali može i brže.



Svega za 3 koraka 8 procesora posjeduju našu informaciju.

U memoriji rezervišemo niz B[j]. Tu će biti smještena kopija podataka koja je namijenjena procesoru p_j. Algoritam prvo svoju informaciju smješta na lokaciju B[0], pa B[1], pa onda paralelno u sledećem koraku B[2] i B[3].

Algoritam: for k=0 to $\lceil \log p \rceil - 1$ do (za osnovu 2)

for $\forall$ procesor j paralelno do
B[$j+2^k$] $\leftarrow$ B[j]
end-for

} unutraš. petlja zahtjeva $o(1)$ vremena

Složenost ovog algoritma je $o(\log p)$. Ako je $j+2^k > p$ takav korak ne izvršavamo, jer nema mo poziciju gdje bi smjestili ovakav podatak.

Pošto naši procesori rade paralelno $B_0 \to B_1$, $B_1 \to B_2$ što je besmisleno, jer B_1 nema pravu informaciju.

Ove korake ne treba izvršavati. Kada je k=0 prelazi na sledeću poziciju. ($\forall b_j \to b_{j+1}$)

k=1 → udvostručavamo korak $b_0 \to b_2$

k=log p → popunjavamo p-ti element

1° korak: procesor p_i upisuje podatak u B[0]
(postavlja promjen. na 1) S ← 1

while S<p do
for $\forall$ proc. j $0 \le j \le \min(S, p-S)$ do
B[j+S] ← B[j]
S ← 2 · S
end-for

} $o(1)$

end-while

for $\forall$ procesor j do
p_j čita B[j]

} $o(1)$, jer se izvršava paralelno

Složenost ovakvog algoritma je $o(\log p)$. Da li se može brže uraditi?

Ne može jer nije dozvoljeno konkurentno čitanje, a mi od k podataka stvaramo 2k.

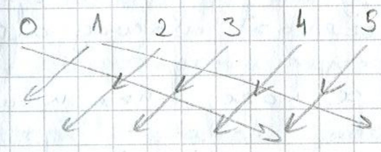
Složenost ovog problema je $\Theta(\log p)$.

Kako da realizujemo na EREW mašini svaki-svakom?

Ako dozvolimo CRCW, mi to realizujemo za $o(\log p)$ vremena, a to isto važi i za ovaj slučaj. Imamo jedan niz B[j], j=0,...,p-1 procesora. Prvo procesor 0 prosljeđi 1, odnosno $\forall$ sledbeniku se prosleđuje svoja vrijednost 1→2, 2→3 (Ciklično prosleđivanje podataka)

Algoritam:

for ∀ procesor j do
P_j upisuje svoj podatak u B[j] } o(1)
for k = 1 to p-1 do → izvršava se sa složenošću O(p)
for ∀ procesor j do
P_j čita podatak B[(j+k) mod p] } o(1)
end for
složenost O(p)



Algoritam sortiranja za

širenje podataka (EREW alg.)

Niz sortiramo ako imamo elemente $S_0 \dots S_{n-1}$. Prvo računamo poredak ∀ element, tj. poziciju elementa gdje se nalazi tako što gledamo koliko elemenata je manje od njega.

for ∀ procesor j do R[j] ← 0
 for k=1 to p-1 do
  for ∀ proc. j do
   l ← (j+k) mod p
o(1) if (S[l] < S[j]) or
   (S[l] = S[j] and l < j)
   then R[j] ← R[j]+1
  endif
 endfor
endfor
for ∀ proc. j do S[R[j]] ← S[j]

Algoritam selekcije

S[j] i dat je broj k. Treba naći k-ti po veličini br. u nizu S.

Ako bi se tražio prvi po veličini, to bi bio min, a k-ti max. Ako nam treba srednji, kako njega odrediti?

Imamo 1 niz i podijelimo ga na 2 i biramo elem.

$\underset{\substack{m \\ \text{pivot}}}{\overline{\quad L \quad | \quad G \quad}}$

Grupišemo elemente manje od pivota i veće od pivota. Ako je pivot na m poziciji i k < m mi nastavljamo traženje k-tog elementa u prvoj polovini, tj. u L i time smo eliminisali 1/2 naših elemenata. Ako je k > m, tražimo (k-m)-ti element u nizu G. Ako bi pivot uvijek bio na sredini, onda bi složenost našeg algoritma zapisali kao $T(n) = T\left(\frac{n}{2}\right) + O(n) \Rightarrow$

Za $O(n)$ vremena određujemo koji je element $>$ a koji je manji.

$\Rightarrow T(n) = O(n)$ (pod pretpostavkom da uvijek biramo u nizu S k-ti po veličini)

Select (S, k)

1. if $|S| < g$ → unaprijed zadat br.
   then sort(S) i return S[k]
   else
   $O(n)$ { podijeli S u $|S|/g$ nizova veličine g; sortiraj ih i od njihovih medijana formiraj niz T veličine $\frac{|S|}{g}$ }
2. $m = \text{select}\left(T, \frac{|T|}{2}\right)$ – tražimo medijanu
3. kreiraj 3 podniza
   L: element od S koji su $<$ od m
   E: –II– od S –II– $=$ m
   G: –II– –II– $>$ m
   } $O(n)$ vremena
4. if $|L| \geq k$ then return select (L, k)
   else
   if $|L| + |E| \geq k$
   then return m
   else return select $(G, k - |L| - |E|)$
   end_if
   end-if

Ulazna formula:

$T(|S|) = O(|S|) + T(\max\{|L|, |G|\})$

$|S| = n, \; g > 4$

$T(n) \leq T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + O(n)$

$T(n) = O(n)$

$T(n) = d \cdot n$

$T(n) = T\left(\frac{n}{g}\right) + T\left(\frac{3n}{4}\right) + c \cdot n$

$g = 5$

$d \cdot n = \frac{d}{5} \cdot n + \frac{3n}{4} \cdot d + c \cdot n \;/ \cdot 20$

$d \cdot n = 20 \cdot c \cdot n \Rightarrow d = 20c$

Ako je g veće, onda će konstanta d biti manja, ali se c povećava jer koristimo sortiranje. Složenost algoritma je $T(n) = O(n)$ u najgorem slučaju.

Sekvencijalan problem

Imamo pravu i n tačaka u ravni. Treba povući pravu paralelnu sa pravom tako da pola tačaka bude sa jedne, a pola sa druge strane

$\sigma(n)$

(x_i, y_i)

$Ax + By + C = 0$

(A, B, C)

$p = n^{1-x}$

$x = \frac{1}{2} \quad p = \sqrt{n} \quad \sigma(n^x) = \sigma\left(\frac{n}{p}\right)$

2.11.2006.

Paralelni algoritam (paralelna sel.)

Imamo n elemenata, koristimo $p = n^{1-x}$ procesora pri čemu je $x < 1$, a tu zahtjevamo i da je $x >$

PRAMselect(S, p, k)

1\. if $|S| < 4$

then sort S } $\sigma(1)$

return $S[k]$

else

broadcast $|S|$ svim procesorima $\log p$

$\sigma(1)$ podijeli S u p podnizova $S^{(j)}$ veličine $\frac{|S|}{p}$

$\forall j$ paralelno procesor p_j računa $T_j \leftarrow$ select $\left(S^{(j)}, \frac{|S^{(j)}|}{2}\right) \rightarrow \sigma(|S^{(j)}|)$

end-if

[procesor p_j računa T_j, tako što izvrši select iz niza $S^{(j)}$ njegove medijane]

[broadcast $|S|$ svim procesorima $\Rightarrow$ dali cardin.

S proslijediti svim procesorima treba nam $\log p$.

Prvi procesor uzima $\frac{|S|}{p}$, drugi od pozicije $\frac{2|S|}{p}$, ...

Svaki proc. može da proračuna koji dio niza njemu pripada.]

2\. korak: $m \leftarrow$ PRAMselect$\left(\overset{\text{niz medijana}}{T}, p, \frac{|T|}{2}\right)$

3. (opet medijanu proširujemo svim procesorima)
broadcast m svim procesorima } $o(\log p)$
podijeli S na nizove
L: elementi iz S koji su < od m
E: —||— —||— —||— = m } $o(n^x)$
G: —||— —||— > m

4. if $|L| \geq k$ then return PRAM select(L, p, k)
else
if $|L| + |E| \geq k$ then return m
else return PRAM select(G, p, $\overbrace{k}^{\text{br. proc.}}$ $\underbrace{-|L|-|E|}_{\text{preskače one iz } L \text{ i } E}$)
end_if
end_if

Za 1. korak
Što se tiče $|S^{(i)}| = \frac{|S|}{p} = \frac{n}{n^{1-x}} = n^x$
Ne pretpostavljamo da je $x = 0$, jer dobijamo 1 i u našem koraku ne dominira n^x već $\log p$.
Ukupno za 1. korak treba $o(n^x)$ vremena i dobijamo $\log p = \log n^{1-x}$.
Ako je $x = 1$ dobijamo 1 procesor i to ne bi bio paralelan algoritam.

2. korak: Radimo paralelno. Pozivamo algoritam paralelno. Zavisi od $|T|$.
3. korak: Niz S dijelimo na nizove L, E, G. Treba nam $\log p$. Svaki od procesora zna vrijednost m.
Tri podniza.
$L^{(i)}, E^{(i)}, G^{(i)}$ → svaki procesor to uredi lokalno.
∀ od podnizova $L^{(i)}$ odvija se paralelno.
Svaki od procesora može izračunati svoju dužinu. $|L^{(i)}|$
Za korak 4. je bitna dužina. Moramo izračunati za koje pozicije procesor treba da upiše u Li. Podnizovi nisu iste kardinalnosti. Zna od koje pozicije da upisuje na osnovu paralelnog prefix (algoritam (koji radi logaritamski). Pa nam i tu treba $o(\log p)$ vremena.
Samo za upisivanje:
Treba nam $o(n^x)$ vremena. Svi procesori mogu da upisuju na određene lokacije. Da bi se upisala sva 3 niza, treba n^x koraka, jer je zbir ta 3 niza ustvari $|S^{(i)}|$.

4. korak : $\frac{3n}{4}$

Ukupna složenost može da se izrazi:

$T(n,p) = T(p,p) + T(\frac{3}{4}n, p) + o(n^x)$

zahtjeva 2 koraka — 3 koraka

$T(n^{1-x}, p)$

Rješavanje ove relacije možemo pretvoriti sa-me u relaciju od n.

Dobija se $T(n,p) = o(n^x)$

$Ts(n) = o(n)$

sekvencijalnog algoritma

Onda je ubrzanje : $speed\_up(n,p) = \frac{n}{n^x} = n^{1-x} = p$

Što je i najbolje ubrzanje (= br. procesora)

Efikasnost $E = 1 \Rightarrow$ Dobar algoritam

16.11.2006.

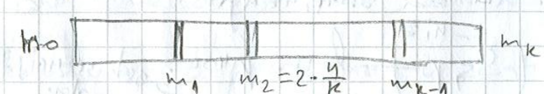
Paralelno sortiranje selekcijom

Imamo neki niz brojeva $x_1, \dots, x_n$ treba da izvršimo takvu permutaciju elemenata da dobijemo $y_1, \dots, y_n$ takvi da je $y_1 \leq \dots \leq y_n$ ($y_i = x_{i_j}$). ∃ različiti paralelni algoritmi za sortiranje. Prvo fiksiramo jedan broj k i izaberemo k−1 element niza. Neka su to elementi $m_1, \dots, m_{k-1}$. Pretpostavićemo da je $m_0 = -\infty$, $m_k = +\infty$ (posmatramo kao brojeve)

element veći od svih naših elemenata

Ovi elementi će vršiti particiju našeg niza.

Pretpostavimo da je $m_i < m_{i+1}$



Ove particije će biti različitih veličina (a mi ćemo nametnuti da budu istih veličina), jer brojeve $m_1, \dots, m_k$ biramo nasumice.

Pretpostavimo da je mi izabran da bude $i \cdot \frac{n}{k}$-ti

niza pomoću m_i, dobićemo k particija koje će biti istih veličina (svaka ima $\frac{n}{k}$ elemenata)

Kad izvršimo particiju pretpostavljamo, da su element lijevo od m_i manji od m_i, a desno od m_i veći od m_i.

Ostaje nam da $\forall$ od particija $[m_i, m_{i+1}]$ sortiramo rekurzivno na isti način i time ćemo dobiti sortirani niz.

Pretpostavka je da imamo $p = n^{1-x}$ procesora $0 < x < 1$ takođe pretpostavljamo da biramo broj k (unaprijed fiksirani broj) koji ćemo izabrati na $k = 2^{1/x}$ (particiju vršimo na k djelova) $x = 1/2$ $p = \sqrt{n}$ $k = 2^2 = 4$

Algoritam za sortiranje

PRAM select Sort (S, p) — niz — br. proc.

1. if $|S| < k$ then return quicksort(S) — sekvencijalni alg.
2. (izbor br. m_i)

   for $i = 1$ to $k-1$ do

   m_i = PRAM Select$(S, i \cdot \frac{|S|}{k}, p)$

   $\{m_0 = -\infty, \; m_k = +\infty\}$

   end_for

3. (vršimo particiju pomoću br. m_i)

   for $i = 0$ to $k-1$ do

   Konstruiši podniz $T^{(i)}$ od elemenata iz S koji su između brojeva m_i i m_{i+1}.

   end_for

   (ostaje nam da na malim particijama izvršimo rekurzivno sortiranje)

4. for $i = 1$ to $k/2$ do in parallel — paralelno sortiranje polovine niza

   PRAM SelectSort$(T^{(i)}, \frac{2p}{k})$

   end_for

   rekurzivno ga pozivamo

   → za jedan podniz koristićemo $\frac{2p}{k}$ procesora, jer istovremeno vršimo sortiranje $k/2$ podnizova, tj. koristimo $k/2$ procesora i za $\forall$ podniz koristimo $\frac{k}{2} \cdot \frac{2p}{k} = p$ procesora

5. (sortiramo podnizove)

   for $i = \frac{k}{2} + 1$ to k do in parallel

   PRAM SelectSort$(T^{(i)}, \frac{2p}{k})$

   end_for

Kardinalnost $|T^{(i)}|$ je povezana sa brojem procesora $|T^{(i)}|^{1-x} = \frac{2p}{k}$ Ako se na početku $|S|^{1-x} = p$ ...

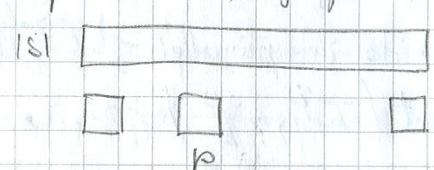
kao br. procesora, sada je

1. korak: nas "košta" konstantno vrijeme $O(1)$

2. korak: k puta pozivamo paralelni select, pa je $k \cdot O(n^x)$

$O(n^x)$ je složenost za drugi korak

3. korak: Imamo niz u zajedničkoj memoriji određeni br. procesora, koji je manji od veličine niza.



Algoritmom broadcasting ćemo proslijediti svim procesorima veličinu S niza. Na osnovu toga i ∀ procesor će proračunati koji dio niza njemu pripada. ∀ procesor uzima $\frac{|S|}{p}$ elemenata i nezavisno vrši particiju svoga dijela.

Jedan procesor posmatra jedan dio niza i na njega vrši particije koje mogu biti ≠ veličina i on računa veličinu svake particije. Primjenićemo k puta paralelni prefix algoritma i dobijamo informaciju od koje do koje pozicije procesor smješta svoje elemente u određenu particiju.

Da bi S proslijedili svim procesorima treba nam $\log p$, a konstantno vrijeme da odredi svoj dio niza.

Procesor ima $\frac{|S|}{p} = \frac{n}{n^{1-x}} = n^x$ elemenata.

Da bi izvršio particiju n-elemenata, treba u $O(n^x)$ vremena.

$$[\log p < O(n^x) \Leftarrow p < n]$$

Koliko nam vremena treba za paralelni prefix algoritma?

Pošto imamo p procesora, potrebno nam je $O(\log p)$ vremena. ∀ procesor treba da prepiše svoje elemente na određene particije za $O(n^x)$ vremena za sve elemente.

Koliko nas "košta" 3. korak? $O(n^x)$

4. i 5. korak imaju istu složenost.

Složenost jednog poziva paralelnog algoritma

$T\left(\frac{n}{k}, \frac{2p}{k}\right)$ $|T^{(i)}| = n/k$

Čitav 4. korak nas "košta" $T\left(\frac{n}{k}, \frac{2p}{k}\right)$

5° Složenost $T(n,p) = \sigma(n^x) + 2\cdot T\left(\frac{n}{k}, \frac{2p}{k}\right)$.

Imamo u vidu da je $p = n^{1-x}$

$$\left(\frac{n}{k}\right)^{1-x} = \left(\frac{n}{2^{1/x}}\right)^{1-x} = n^{1-x}\cdot\frac{1}{2^{\frac{1-x}{x}}} = \frac{2\cdot n^{1-x}}{2^{1/x}} = \frac{2p}{k}$$

Tačno dobijamo vezu između $\frac{n}{k}$ i $\frac{2p}{k}$

Ovakva relacija nam daje:

$T(n,p) = \sigma(n^x \log n)$

$T_s(n) = \sigma(n \log n)$
sekv. alg.

Ubrzanje sekvencijalnog algoritma je

$$\text{speed-up}(n,p) = \frac{n\log n}{n^x \log n} = n^{1-x} = p$$

Ubrzanje je max.

Efikasnost je jednaka 1: $Efficiency = \sigma(1)$

Cijena ili rad $work(n,p) = \sigma(n\log n)$ ima dobre performanse.

Pretpostavka je da je br. procesora manji od dužine niza, inače nećemo imati dobru efikasnost.

Postoji varijanta quick sorta koja uvijek od elemenata niza izabere srednji. U praksi je mnogo efikasnije kad slučajno biramo element. Isto važi i za paralelni algoritam.

Alternativni algoritam za sortiranje

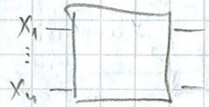
PRAM randomSort(S,p)

1. for $\forall$ procesor j do
   $\frac{|S|}{p^2}$ elemenata od svojih $\frac{|S|}{p}$ elemenata i smjesti na odgovarajuću poziciju niza T
2. procesor po sortira niz T (određuje m_i)
   $m_i = i\cdot\frac{|S|}{p^2}$ -ti element T
3. for $\forall$ procesor j do
   smjesti svoje elemente između m_i i m_{i+1} u $T^{(i)}$
4. for $\forall$ procesor j do
   sort $(T^{(i)})$ i-ti procesor uzme $T^{(i)}$ i sortira ga

Mreže za sortiranje

- to je jedno kombinatorno kolo sa n ulaza i n izlaza pri čemu su izlazi sortirani ulazi.

Može se predstaviti:



y_i permutacije od x_i

i važi $y_1 \leq \ldots \leq y_n$

Pretpostavljamo da je na vrhu min, a na dnu max.

Kakva je mreža 2 elementa? Mreža 2 elem.

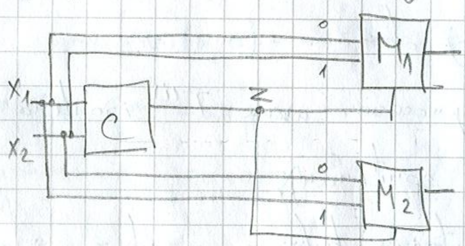
$y_1 = min(x_1, x_2)$
$y_2 = max(x_1, x_2)$

Ovu mrežu za sortiranje pravimo pomoću kombinatornih kola.

Trebaju nam 2 MUX i 1 komparator (daje izlaz 0 ako je $x_2 >$, a izlaz 1 ako je $x_1 >$ kad uporedi x_1 i x_2)

$$z = \begin{cases} 1, & x_1 > x_2 \\ 0, & x_1 \le x_2 \end{cases}$$

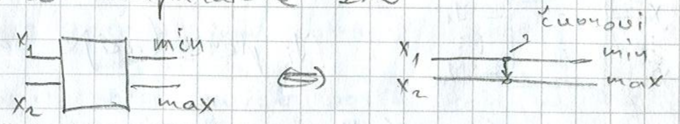
MUX propušta 1 od 2 moguće vrijednosti

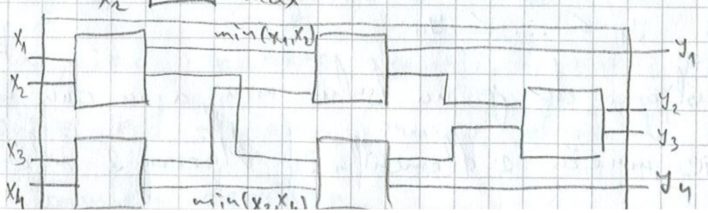


Ako je na ulazu 0 propušta prvu vrijednost

Kako realizovati mrežu sa n ulaza i n izlaza?

- koristimo komparatore 2x2





Prolaz kroz jedan element 2x2 je konstantno vrijeme. Koji je najveći br. elem. kroz koje prođe signal?

cijena = 5 (br. elem. komparatora)

čekanje = 3

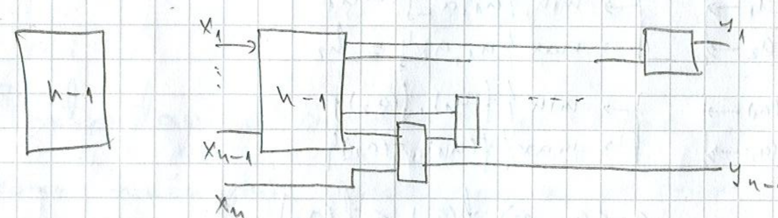
Prvo realizujemo mrežu za n-1 elem., pa naknadno dodamo n-ti element ⊛

(Bazirana je na insert algoritam)

I koliko koristimo elem.

II čekanje (vrijeme za sortiranje)

⊛



n-ti elem. možemo uporediti sa 2 elem.

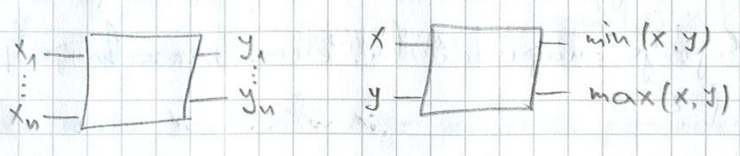
$C(n) = C(n-1) + n - 1 = O(n^2)$

Zadržavanje kroz ovu mrežu je

$$D(n) = D(n-1) + n - 1 = O(n^2)$$

23.11.2006.

Mjera da li koristimo mrežu 2 × 2

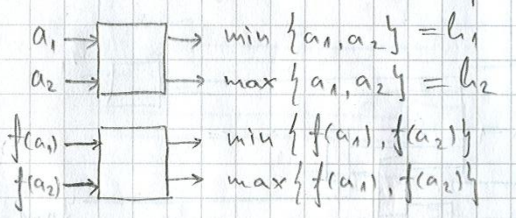


$y_1 \le \ldots \le y_n$

T. Neka je data monotono rastuća f-ja $f^{\uparrow}$, tada ako mreža sastavljena od komparatora ima za ulaz $a_1, \ldots, a_n$ daje izlaz $b_1, \ldots, b_n$ onda će ta mreža za ulaz $f(a_1), \ldots, f(a_n)$ dati izlaz $f(b_1), \ldots, f(b_n)$

D. (indukcijom po dužini)

1° ako imamo samo 1 komparator

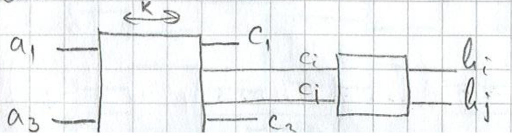


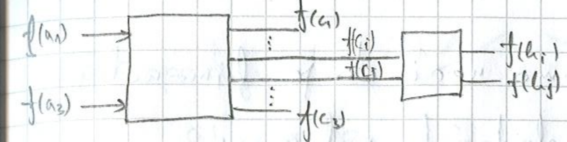
$$a_1 < a_2 \Rightarrow f(a_1) < f(a_2)$$

$$\min\{f(a_1), f(a_2)\} = f(\min\{a_1, a_2\}) = f(b_1)$$

$$\max\{f(a_1), f(a_2)\} = f(\max\{a_1, a_2\}) = f(b_2)$$

Pretpostavimo da ovo važi do dužine k, kažimo da važi za $k+1$.





T. Neka imamo mrežu komparatora veličine $n \times n$ tada ova mreža sortira proizvoljni niz br. $x_1, \ldots, x_n$ ako za sve moguće nizove 0 i 1 dužine n mreža ih sortira.

$$\Big(\forall \underbrace{(y_1, \ldots, y_n)}_{2^n},\ y_i \in \{0,1\}\Big) \text{ mreža će sortirati ovaj niz}$$

Ova teorema se zove 0-1 teorema jer proizvoljni niz prevodimo u 0 i 1.

D: $\Rightarrow$ Ako sortira proizvoljni niz sortiraće i niz 0 i 1.

$\Leftarrow$ Pretpostavimo suprotno tj da $\exists$ niz $x_1, \ldots, x_n$ na kome mreža daje izlaz $a_1, \ldots, a_n$ pri čemu $\exists i$ tako da je $a_i > a_{i+1}$. Konstruišimo niz x_j koji je jednak

$$z_j = f(x_j) = \begin{cases} 0, & x_j < a_i \\ 1, & x_j \ge a_i \end{cases}$$

da je ova f-ja $f^{\uparrow}$ definisana na ovaj način. Tada za ulaz z_j dobićemo i izlaz $f(a_j)$

$z_j = f(x_j) \Rightarrow b_j = f(a_j)$ pri tome važi:

$$b_i = f(a_i) > f(a_{i+1}) = b_{i+1}$$

Za niz $x_1, \dots, x_n$ ćemo reći da je bimonoton ako važi jedan od sledećih uslova:

1) $\exists i: \; x_1 \le \dots \le x_i$

$x_{i+1} \ge x_{i+2} \ge \dots \ge x_n$

i

2) $\exists i: \; x_1 \ge \dots \ge x_i$

$x_{i+1} \le x_{i+2} \le \dots \le x_n$

3) Ako cikličnom permutacijom naših elem. možemo dobiti da važi 1 ili 2

Pr. 15, 11, 3, 5, 7, 18, 16 da li je ovaj niz bimonoton

Jeste, jer cikličnom permutacijom utvrđujemo da raste do 18 pa opet pada

$\underline{3, 5, 7}, 18, \underline{16, 15, 11}$

Interesuje nas da napravimo šemu koja će vršiti sortiranje niza koji je bimonoton. Zamislimo da je $a_0, \dots, a_n$, $n = 2^k$ bitonon niz

Definišimo niz $b_i = \min\{a_i, \frac{a_n}{2} + i\}$ i

$b_{\frac{n}{2}+i} = \max\{a_i, a_{\frac{n}{2}}\}$, pri čemu je $i < \frac{n}{2}$, $n = 2^k$

Ako pretpostavimo da je $a_0 \le \dots \le a_{\frac{n}{2}-1}$

$a_{\frac{n}{2}} \ge a_{\frac{n}{2}+1} \ge \dots \ge a_{n-1}$

U jednom pravcu elem. rastu, a u drugom dijelu niza opadaju

$\exists i: \quad a_{i-1} < a_{\frac{n}{2}+i-1}$

$a_i > a_{\frac{n}{2}+i}$ tada baš kod a_i dolazi do prelamanja niza

$$\left.\begin{aligned} &b_0 \le \dots \le b_{i-1} \\ &b_i \ge b_{i+1} \ge \dots \ge b_{\frac{n}{2}-1} \end{aligned}\right\}$$

do i-tog b raste, a posle i-tog opada

$b_i = \max\{b_j\}$, $0 \le j \le \frac{n}{2}-1$

$b_{\frac{n}{2}} \ge b_{\frac{n}{2}+1} \ge \dots \ge b_{\frac{n}{2}+i-1}$

$b_{\frac{n}{2}+i} \le b_{\frac{n}{2}+i+1} \le \dots \le b_{n-1}$

$b_{\frac{n}{2}+i}$ je $\min\{b_j\}$, $\frac{n}{2} \le j \le n-1$

$b_i \le b_{\frac{n}{2}+i} = \max\{a_i, a_{\frac{n}{2}+i}$

$\min\{a_i, a_{\frac{n}{2}+i}\}$

Svi ele. u prvoj polovini su manji od elem. u drugoj polovini.

$b_{j_1} \le b_{j_2}$

$j_1 < \frac{n}{2}$

$j_2 \ge \frac{n}{2}$