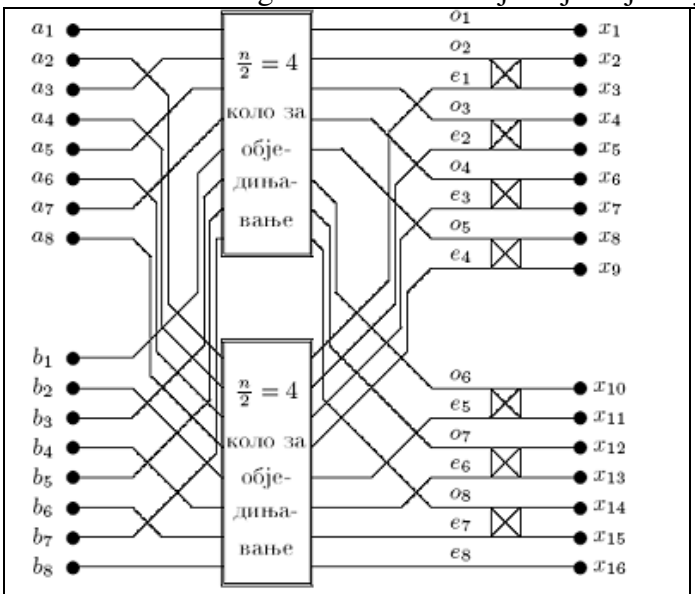
Paralelni algoritmi za mreže računara

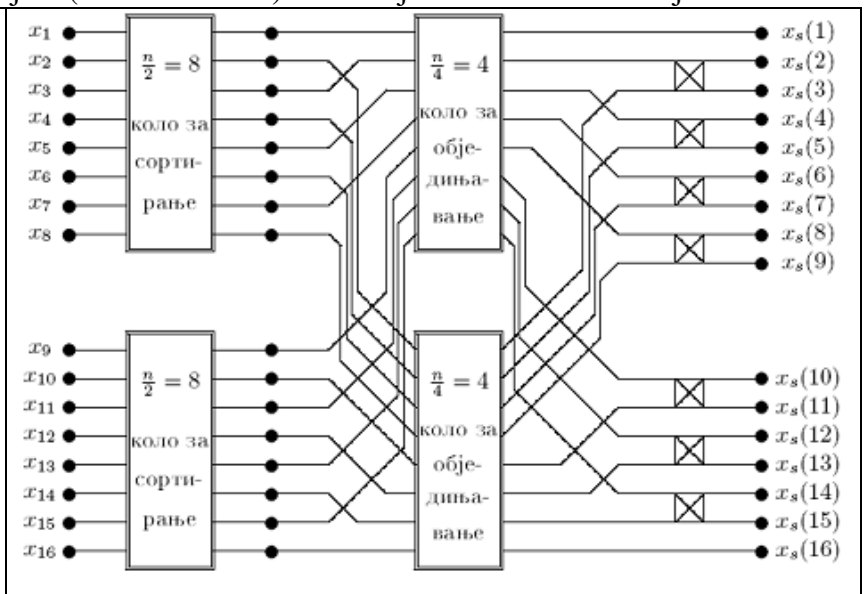
1. Dato je n procesora $P_1, P_2, \ldots, P_n$ tako da procesor P_i čuva vrednost x_i. Cilj je preraspodeliti brojeve tako da najmanji broj bude u P_1, sledeći u P_2 itd. Procesori su povezani u linearni niz, tj. procesor P_i povezan je sa procesorom P_{i+1}, $i = 1, \ldots, n-1$.
Rešenje:
Sortiranje parno-neparnim transpozicijama
Algoritam Sortiranje_na_nizu(x, n);
Ulaz: x (vektor sa n elemenata, pri čemu je x_i u procesoru P_i).
Izlaz: x (sortirani vektor, tako da je i-ti najmanji element u P_i).
begin
do in parallel $\lceil n/2 \rceil$ puta
P_{2i-1} i P_{2i} upoređuju svoje elemente i po potrebi ih razmenjuju;
{za sve i, takve da je $1 < 2i <= n$}
P_{2i} i P_{2i+1} upoređuju svoje elemente i po potrebi ih razmenjuju;
{za sve i, takve da je $1 <= 2i < n$ }
{ako je n neparno, ovaj korak se izvršava samo $\lfloor n/2 \rfloor$ puta }
end

Ideja je da svaki procesor uporedi svoj broj sa brojem jednog svog suseda i razmenjuje vrednosti ako su brojevi pogrešno uređeni, a zatim isti posao obavlja sa svojim drugim susedom, i nstavlja se ovaj proces dok se ne dobije željeni raspored.
Koraci se dele na
neparne (u kojima procesori sa neparnim indeksom vrše poređenje sa svojim desnim susedom)
parne (u kojima procesori sa parnim indeksom vrše poređenje sa svojim desnim susedom)

2. Paralelizovati algoritam za sortiranje objedinjavanjem (MERGESORT) korišćenjem mreža sa sortiranje.



Слика 1



Слика 2

Kod konstrukcije efikasnih sekvencijalnih algoritama od interesa nam je ukupan broj koraka. Kod konstrukcije paralelnih algoritama cilj je da ti koraci budu u što većoj meri nezavisni. Jedan od primera za nezavisne korake, jeste sortiranje objedinjavanjem, jer su rekurzivni pozivi potpuno nezavisni i mogu da se izvršavaju paralelno. Međutim, kada želimo da objedinimo dva dobijena sortirana podniza, to moramo da uradimo na sekvencijalni način - i+1-vi član se stavlja u vektor tek pošto je tamo stavljen i-ti član.

Nešto drugačiji algoritam se zasniva na dekompoziciji. Zbog jedostavnosti, pretpostavimo da je n stepen dvojke, i neka su $a_1, a_2, \ldots, a_n$; $b_1, b_2, \ldots, b_n$ dva sortirana niza koje treba objediniti.
Rezultat objedinjavanja biće niz $x_1, x_2, \ldots, x_{2n}$.
Znamo da je sigurno $x_1 = \min\{a_1, b_1\}$, međutim različite delove ovih podnizova treba objediniti paralelno. To se postiže podelom ova dva niza na po dva dela – sa neparnim i parnim indeksima. Svaki od delova se objedinjava sa odgovarajućim delom drugog niza i na kraju se kompletira objedinjavanje.

Neka je $o_1, o_2, \ldots, o_n$, objedinjeni redosled neparnih nizova $a_1, a_3, \ldots, a_{n-1}$; $b_1, b_3, \ldots, b_{n-1}$,
Neka je $e_1, e_2, \ldots, e_n$, objedinjeni redosled parnih nizova $a_2, a_4, \ldots, a_n$; $b_2, b_4, \ldots, b_n$.
Po konstrukciji važi da je $x_1=o_1$ i $x_{2n}=e_n$.
Ostatak objedinjavanja izvodi se na osnovu sledeće teoreme:
Teorema: U skladu sa uvedenim oznakama, za i = 1, 2, ... , n-1 važi $x_{2i} = \min\{o_{i+1}, e_i\}$, $x_{2i+1} = \max\{o_{i+1}, e_i\}$.
Važna posledica ove teoreme je da se objedinjavanje nizova $o_1, o_2, \ldots, o_n$ i $e_1, e_2, \ldots, e_n$, obavlja u jednom paralelnom koraku.
Na slici 1 data je rekurzivna skica objedinjavanja.
Na slici 2 prikazano je kompletno sortiranje - sortiranje parno-neparnim objedinjavanjem.

3. Mreža računara zadata je kao kompletno binarno stablo visine $h - 1$ sa $n = 2^{h-1}$ listova, odnosno ukupno 2^h-1 procesora pridruženih čvorovima stabla. Ulaz je niz $x_1, x_2, \ldots, x_n$ tako da se x_i čuva u listu i. Treba odrediti k-ti najveći element.
Pogledati poglavlje 12.4.3 u udžbeniku.

4. Kako treba modifikovati algoritam za nalaženje k-tog najvećeg broja među brojevima $x_1, x_2, \ldots, x_n$ na stablu, tako da radi ispravno i ako brojevi $x_1, x_2, \ldots, x_n$ nisu svi različiti.
Rešenje:
Jedna iteracija procesa selekcije, kao i u osnovnoj verziji, sastoji se od četiri faze, tj. četiri prolaska stabla (gore,dole,gore,dole).
1. U prvoj fazi se bira pivot, koji dospeva u koren stabla.
2. U drugoj fazi se pivot dostavlja naniže svim (aktivnim) listovima, isto kao i u osnovnoj verziji.
3. U trećoj fazi svaki list upoređuje svoj broj sa pivotom i šalje naviše rezultat poređenja, ali ovog puta precizniji - da li je veći od pivota, odnosno da li je jednak pivotu. Na putu do korena stabla se tako dobija broj v listova većih od pivota i broj j listova jednakih pivotu.
4. U četvrtoj fazi se ova dva broja dostavljaju svim aktivnim listovima. Tada svaki aktivni list upoređuje k sa v i $v + j$, pa ako je
- $k <= v$, onda se list iskljucuje, ako je njegov broj manji ili jednak od pivota, a medju neiskljucenim listovima u narednoj iteraciji trazi se k-ti najveci,
- $v < k <= v + j$, onda se list iskljucuje, ako je njegov broj manji ili veci od pivota, a medju neiskljucenim listovima u narednoj iteraciji trazi se $(k-v)$-ti najveci,
- $k > v + j$, onda se list iskljucuje, ako je njegov broj veci ili jednak od pivota, a medju neiskljucenim listovima u narednoj iteraciji trazi se $(k - v - j)$-ti najveci.

5. Na mreži računara u obliku binarnog stabla, pri čemu se u i-tom listu nalazi broj x_i, $1 <= i <= n$, treba rešiti problem paralelnog prefiksa, tj. nakon izvršenja algoritma list i treba da sadrži sumu $x_1 * x_2 * \ldots * x_i$, gde je * asocijativna operacija. Vreme izvršavanja treba da bude $O(\log n)$.

Konstruišimo rešenje koristeći indukciju.
Baza: Ako je visina stabla (broj slojeva procesora) 2, onda levi list šalje svoj broj korenu, koji ga zatim šalje naniže desnom listu, a koji ga dodaje svom broju x_2.
Induktivna hipoteza: Pretpostavimo da imamo algoritam za visinu $h >= 2$.
Induktivni korak: Razmotrimo stablo visine $h + 1$ (razmatramo samo slučaj kompletnih binarnih stabala; algoritam se lako prepravi da radi u opštem slučaju).
Neka je R koren, i neka su R_L i R_D njegov levi i desni sin. Sumu svih listova podstabla zvaćemo *sumom tog podstabla*.
Najjednostavnije je rešiti problem nezavisno za levo i desno podstablo, a zatim preko korena poslati sumu levog podstabla svim listovima u desnom podstablu. Svaki list desnog podstabla prosto dodaje sumu levog podstabla izračunatom prefiksu.
Problem sa ovim rešenjem je u tome, što mu je vreme izvršavanja $O(h^2)$, zbog diferencne jednačine
$T(h + 1) = T(h) + h$
(ako je $T(h)$ vreme izvršavanja na stablu visine h).
Ovo resenje moze se popraviti, ako zapazimo da nema potrebe cekati da levo stablo zavrsi izracunavanje. Desno podstablo treba da dobije sumu levog podstabla, a ta suma moze biti raspoloziva u korenu u koraku $h + 1$.

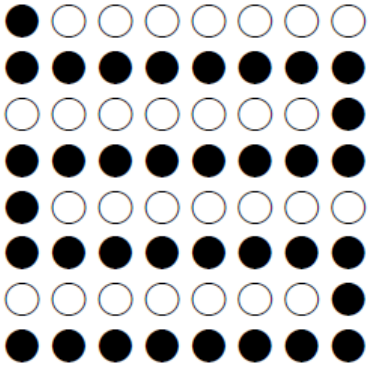
Dakle, zahtev koji R_L treba da ispuni je da primi sumu svog podstabla i posalje je navise ka korenu.
Zahtev za R_D je da tu sumu primi od korena i pošalje je naniže svojim potomcima.
Zaključujemo da treba da važe sledeća pravila:

- svaki list počinje slanjem svog broja naviše (treba dakle promeniti prosto rešenje za stablo visine 2, jer koren treba da zna sumu);
- unutrašnji čvor, kad dobije vrednosti od sinova, sabira ih i šalje naviše;
- unutrašnji čvor, kad dobije vrednost od oca, šalje tu vrednost i levom i desnom sinu;
- unutrašnji čvorovi takođe igraju ulogu korena svojih podstabala, i vrednost dobijenu od levog sina šalju desnom sinu.

Vreme izvršavanja algoritma: $2h$.

6. Dat je kvadratna matrica (*meš*) od $n \times n$ procesora od kojih svaki sadrži po jedan piksel $n \times n$ crno-bele slike. Drugim rečima, svaki procesor sadrži po jednu binarnu cifru, gde 1 odgovara crnoj boji, a 0 beloj. Potrebno je naći *povezane komponente* slike. Dva crna piksela pripadaju istoj komponenti ako postoji crna staza koja ih povezuje (horizontalno ili vertikalno; dijagonale se ne smatraju vezom). Svi pikseli u istoj komponenti treba da budu označeni jedinstvenom oznakom komponente. Inicijalno, piksel (ij) je označen vrednošću $in + j$ $(i, j = 0, 1, \ldots, n - 1)$. Razmatramo sledeći algoritam koji označava svaku komponentu najmanjom oznakom piksela koji joj pripadaju. U svakom koraku, svaki procesor koji sadrži piksel crne boje proverava oznake susednih piksela. Ako bilo koji od njih ima manju oznaku, onda procesor tu vrednost upisuje kao oznaku piksela koji mu je pridružen. Dokazati da će opisani algoritam u konačnom broju koraka označiti sve komponente korektno, ali i da to može da zahteva cn^2 paralelnih koraka za neki ulaz (c je konstanta). Kako detektovati kada algoritam treba da prekine svoj rad?

Rešenje:
Kriterijum za prestanak izvršavanja može da bude ispunjen uslov da u koraku nije načinjena nijedna izmena oznake. Na početku izvršavanja algoritma, zbir svih oznaka je konačan nenegativan broj, pa kako se u svakom koraku taj zbir smanjuje (ali ostaje nenegativan), sledi da algoritam završava rad u konačnom broju koraka.
Na osnovu opisa algoritma jednostavno se dokazuje da su na kraju njegovog izvršavanja svi pikseli jedne komponente označeni kao i piksel sa najmanjom oznakom na početku. Takodje, jednostavno se pokazuje da beli pikseli ne mogu promeniti oznaku i da pikseli jedne komponente ne mogu dobiti oznaku piksela iz neke druge komponente.
Algoritam može da završi rad u linearnom vremenu (ako su svi pikseli označeni), ali i za cn^2 paralelnih koraka, što ilustruje sledeće početno stanje mreže za $n = 8$:



Za mrežu dimenzije $n \times n$, za analogno početno stanje, postoji samo jedna povezana komponenta i u svakom koraku može da samo jedan crni piksel dobije potrebnu vrednost. Njihov broj je $n*n/2 + n/2 = n(n+1)/2$ odnosno asimptotski cn^2.