Paralelni algoritmi

1. Neka je * proizvoljna asocijativna binarna operacija (tj. važi: $x * (y * z) = (x * y) * z$ za proizvoljne x, y, z) i neka je dat niz brojeva $x_1, x_2, \ldots, x_n$.
Izračunati proizvode $x_1 * x_2 * \ldots * x_k$ za $k = 1, 2, \ldots, n$.
Ovaj problem nazivamo paralelni problem prefiksa.
Operacija * može, na primer, biti: sabiranje, množenje, max dva broja.

Sekvencijalna verzija problema prefiksa je trivijalna, jer prefiksi se jednostavno izračunavaju redom. Paralelni problem prefiksa nije tako lako rešiti. Iskoristićemo metod dekompozicije, uz pretpostavku da je n stepen dvojke.
Označimo sa $PR(i, j)$ proizvod $x_i * x_{i+1} * .. * x_j$. Potrebno je izračunati $PR(1, k)$ za $k = 1, 2, ..., n$.
Vrednosti $PR(1,m)$ za $m = n/2+1, n/2+2, ...; n$ dobijaju se izračunavanjem proizvoda $PR(1, n/2) * PR(n/2+1,m)$.

Oba potproblema se mogu rešiti paralelno u modelu EREW. Ako imamo n procesora za problem veličine n, onda se polovina njih može dodeliti svakom potproblemu. Kombinovanje rešenja potproblema sastoji se od $n/2$ množenja, koja se mogu izvršiti paralelno, ali je potreban model CREW, jer se u svakom množenju koristi $PR(1, n/2)$ odnosno $x[Srednji]$. Iako više procesora istovremeno čitaju $x[Srednji]$, oni pišu na različite lokacije, pa model CRCW nije neophodan.

```
Algoritam Paralelni_prefiks_1(x, n)
Ulaz x (niz od n elemenata)
Izlaz x (niz čiji i-ti element sadrži i-ti prefiks)
begin
    PP_1(1, n);
end
Procedure PP_1(Levi, Desni)
begin
    if Desni − Levi = 1 then
        x[Desni] := x[Levi] * x[Desni];
    else
        Srednji := (Levi + Desni − 1)/2;
        do in parallel
            PP_1(Levi, Srednji);
            PP_1(Srednji + 1, Desni);
        for i := Srednji + 1 to Desni do in parallel
            x[i] := x[Srednji] * x[i];
end
```

Ukupan broj koraka je $T(n, n) = O(\log n)$, pa je efikasnost algoritma $E(n, n) = O(1/\log n)$ (vreme izvršavanja sekvencijalnog algoritma je $O(n)$).
Na žalost, efikasnost ovog algoritma ne može se poboljšati korišćenjem Brentove leme, jer je ukupan broj koraka na svim procesorima $O(n \log n)$. Prema tome, da bi se poboljšala efikasnost, mora se smanjiti ukupan broj koraka.

```
Algoritam Paralelni_prefiks_2(x, n)
Ulaz x (niz od n elemenata)
Izlaz x (niz čiji i-ti element sadrži i-ti prefiks)
begin
    PP_2(1);
end
Procedure PP_2(Korak)
begin
    if Korak = n/2 then
        x[n] := x[n/2] * x[n];
    else
        for i := 1 to n/(2 * Korak) do in parallel
            x[2 * i * Korak] := x[(2 * i - 1) * Korak] * x[2 * i * Korak];
        PP_2(2 * Korak);
        for i := 1 to n/(2 * Korak) - 1 do in parallel
            x[(2*i+1)*Korak] := x[2*i*Korak]*x[(2*i+1)*Korak];
end
```

Ideja koja omogućuje rešavanje problema je korišćenje iste induktivne hipoteze kao i u prethodnom algoritmu, ali uz podelu ulaza na drugačiji naqin. Pretpostavimo ponovo da je n stepen dvojke i da imamo n procesora.
Neka E označava skup svih x_i sa parnim indeksima i. Ako izračunamo prefikse svih elemenata iz E, onda je izračunavanje ostalih prefiksa (onih sa neparnim indeksima) lako:
ako je poznato $PR(1, 2i)$, onda se neparni prefiks $PR(1, 2i+1)$ dobija izračunavanjem samo još jednog proizvoda $PR(1, 2i) * x_{2i+1}$, $i = 1, 2, \ldots, n/2$
Prefiksi elemenata iz E mogu se odrediti u dve faze.
Najpre se (paralelno) izračunavaju proizvodi $x_{2i-1} * x_{2i}$, koji se zatim smeštaju u x_{2i},
Drugim rečima, izraqunavaju se proizvodi svih elemenata iz E sa svojim levim susedima. Zatim se rešava (indukcijom) problem paralelnog prefiksa za $n/2$ elemenata iz E.
Rezultat za svako x_{2i} je tačan prefiks, jer je svako x_{2i} već zamenjeno proizvodom sa $x2i$-1.
Pošto se znaju prefiksi za sve elemente sa parnim indeksima, preostali prefiksi se mogu izračunati u jednom paralelnom koraku na već spomenuti način.

Lako se proverava da se ovaj algoritam može izvršavati u modelu EREW.

Po Brentovoj lemi, efikasnost je O(1).

2.

Dat je niz od n memorijskih lokacija i n procesora ($n = 2^k$) koji funkcionišu po $EREW$ modelu. Lokacija $A[i]$ sadrži vrednost a_i, $1 \leq i \leq n$. Opisati algoritam sa vremenom izvršavanja $O(\log n)$ nakon čijeg izvršavanja lokacija $A[i]$ sadrži vrednost $\prod_{k=1}^{i} a_k$, za $1 \leq i \leq n$.

3.

Naći paralelni algoritam za izračunavanje vrednosti polinoma $a_0 + a_1x + \ldots + a_nx^n$ sa vremenom izvršavanja O(log n) na računaru sa n procesora koji funkcioniše po EREW modelu ($n = 2^k$). Koeficijenti su smesteni u vektor duzine n + 1 u zajedničkoj memoriji.

4.

Kronekerov proizvod vektora $A[0], A[1], \ldots, A[n-1]$ i $B[0], B[1], \ldots, B[m-1]$ se definiše kao vektor $C[0], C[1], \ldots, C[nm-1]$ sa elementima:

$$C[km + r] = A[k]B[r],\ 0 \leq k \leq n-1,\ 0 \leq r \leq m-1.$$

Drugim rečima C sadrži $n \times m$ matricu čiji je (i, j)-ti element $A[i]B[j]$.

Konstruisati paralelni EREW algoritam za izračunavanje vektora C pomoću p procesora za vreme $O(mn \log p/p)$. Na raspolaganju je memorijski prostor veličine $O(mn)$ i može se pretpostaviti da su m, n i p stepeni dvojke.
Rešenje:

Rešenje problema se može podeliti u dva dela.
Prvi deo je "proširivanje" vektora A, B u vektore matrice A', B' duzine nm,
tako da
$A'[km+r] = A[k]$,
$B'[km+r] = B[r]$

Ovo je neophodno da bi se izbeglo istovremeno citanje istog elementa vektora od strane vise procesora).

Drugi deo je paralelno mnozene $C[i] = A'[i]B'[i]$, $0 <= i <= nm-1$.
Prosirivanje se sa p paralelnih procesora moze izvrsiti za $O(\log p)$ koraka u okviru jedne od mn/p grupa po p elemenata, pa je trajanje prve faze $O(mn \log p/p)$ (pretpostavlja se da je $p <= mn$).
Mnozenje se izvrsava za $O(mn/p)$ paralelnih koraka (obrada svake od mn/p grupa izvrsava se za jedan paralelni korak).

5. Data je povezana lista od n elemenata koji su smesteni u niz A duzine n (redosled elemenata liste nezavisan je od redosleda u nizu). Rang elementa u povezanoj listi definise se kao rastojanje elementa od kraja liste (prvi element ima rang n, drugi $n - 1, \ldots$). Izracunati rangove svih elemenata liste.

```
Algoritam Rangovi(N)
Ulaz N (niz od n elemenata)
Izlaz R (rangovi svih elemenata u nizu)
begin
    D := 1;
    do in parallel
        R[i] := 0;
        if (N[i] = NIL) then R[i] := 1;
        while (R[i] = 0) do
            if (R[N[i]] ≠ 0) then
                R[i] := D + R[N[i]];
            else
                N[i] := N[N[i]];
                D := D * 2;
end
```

6. Dokazati da je za izvršavanje algoritma *Rangovi* dovoljan model EREW.
Rešenje:
Dovoljno je da svaki procesor P_i izračunava svoju sopstvenu kopiju $D[i]$ promenljive D.
Dakle, inicijalizacija je paralelni korak $D[i] := 1$, a ostala pojavljivanja
D zamenjuju se sa $D[i]$.
Drugih konflikata pri pristupanju memoriji nema:
samo procesor P_i čita $R[N[i]]$ (rang svog suseda) i eventualno poziciju njegovog suseda $N[N[i]]$;
procesori čijim se elementima pronađe rang, isključuju se i ne izazivaju konflikte.

7. Data je povezana lista čiji su elementi (proizvoljnim redosledom) smešteni u vektor. Konstruisati efikasan paralalni algoritam za formiranje povezane liste od istih elemenata u obrnutom redosledu (ne premeštajući nijedan element). Može se pretpostaviti da je na raspolaganju dovoljan memorijski prostor za dopunske pokazivače.
Rešenje:
Ako se svakom slogu pridruži procesor, onda u jednom paralelnom koraku svaki procesor upisuje indeks (adresu) svog sloga u odgovarajuće polje sledećeg sloga u listi.
Nakon tog, svaki procesor zna svog prethodnika u listi.

8.
U CRCW modelu odrediti najmanji od n brojeva datih u nizu tako da vreme izvršavanja algoritma bude $O(1)$ i da on koristi najviše n^2 procesora.
Resenje:

Za n datih elemenata postoji $\frac{n(n-1)}{2}$ mogućih parova, pa je $\frac{n(n-1)}{2}$ procesora dovoljno za odredjivanje najmanjeg elementa u konstantnom vremenu primenom sledećeg algoritma: Paru (x_i, x_j) $i < j$, pridružujemo procesor P_{ij} i svakom elementu x_i pridružujemo deljenu promenljivu v_i, tako da je na početku $v_i = 1$ za svako i.

U prvom koraku svaki procesor poredi "svoja" dva elementa i zapisuje 0 u deljenu promenljivu pridruženu većem elementu. Pošto je samo jedan element manji od svih ostalih, samo jedan v_i zadržava vrednost 1. Pretpostavka je da su svi elementi niza različiti.

U sledećem koraku detektuje se kom elementu odgovara jedino v_i čija je vrednost 1. To se može izvesti tako što će procesori pridruženi tom elementu upisati njegovu vrednost u neku zajedničku promenljivu *rezultat*, i ta vrednost je vrednost minimalnog elementa datog niza.

Vreme izvršavanja algoritma je $O(1)$.