**1 PRAM**

1.1 Napisati algoritam za CRCW/EREW PRAM računalo koji će za zadano polje P[] provjeriti ima li u polju elemenata jednakih vrijednosti (duplikata). Za polje od n elemenata na raspolaganju je n procesora. Rezultat mora biti zapisan u jednoj izlaznoj varijabli. Ocijeniti složenost algoritma.

Ideja za CRCW:

U prvoj iteraciji se uspoređuje prvi član sa članovima od 2 do n-tog. U drugom koraku se uspoređuje drugi član sa članovima od 3 do n-tog...

Rez = 0;

for(i=0; i < n-2; i++) {

paralelno(j = i+1; j < n -1; j++) {

if(P[ j ] == P[ i ])

Rez = 1;

}

}

Druga ideja za CRCW:

U prvom koraku se uspoređuje prvi član s drugim, drugi s trećim, treći s četvrtim… U drugom koraku se uspoređuje prvi član s trećim, drugi s četvrtim, treći s petim…

Rez = 0;

for(i = 1; i < n; i++) {

paralelno(j = 0; j < n - i; j++) {

if(P[ j ] == P[ j+i ])

Rez = 1;

}

}

Ideja za EREW:

Sa n procesa u O(1) moguće je kopirati polje P u neko polje K. Dodatno stvaramo polje DUPLIKATI. Na indexu i

u polju DUPLIKATI piše 1 ukoliko je element u polju P indeksiran sa

i duplikat. Inače piše 0.

Ostatak koda je sličan kao druga ideja za CRCW

paralelno(i = 0; i < n; i++) {

K[ i ] = P[ i ];

DUPLIKATI[ i ] = 0;

}

for( i = 1; i < n ; i++) {

paralelno(j = 0; j < n - i; j ++) {

if( P[ j ] == K[ j+i ])

DUPLIKATI[ j ] = 1;

}

}

Rezultat = OR\_REDUCE(DUPL[]);

složenost O(n)

U EREW primjeru moguće je izvesti petlju tako da ima manje od n iteracija (oko n/2, za informacije pogledati video sa PRAM primjeri).

1.2 Napisati algoritam za CRCW/EREW PRAM računalo koji će za zadano polje P[] odrediti broj različitih vrijednosti elemenata polja. Npr. za polje [1, 2, 1, 3, 4, 2, 5, 1] rezultat iznosi 5. Za polje od n elemenata na raspolaganju je n procesora. Rezultat mora biti zapisan u jednoj izlaznoj varijabli. Ocijeniti složenost algoritma.

Ideja za CRCW: Napraviti dodatno polje UNIQUE čija je veličina jednaka polju P i gdje su svi elementi jednaki 1 ( O(1) ). Nakon toga se ulazi u petlju. U prvom koraku petlje svi procesori uspoređuju neku vrijednost polja P (ovisno o dobivenom indeksu) sa prvim članom. Ako su jednaki, na lokaciju UNIQUE[index] upisati 0. Konačno potrebno je napraviti +\_reduce.

paralelno(i = 0; i < n; i++) {

UNIQUE[ i ] = 1;

}

for(i = 0; i < n; i++) {

paralelno( j = i + 1; j < n; j++) {

if(P[ i ] == P [ j ])

UNIQUE[ j ] = 0;

}

}

Rez = +\_reduce(UNIQUE []);

složenost O(n)

Ideja za EREW PRAM: Polje P[] se kopira u polje K[] (O(1)). Slično kao u CRCW mogu se raditi usporedbe, te ako nešto nije jedinstveno zapisuje se 0 u UNIQUE[] na odgovarajućem indeksu. Konačno, napravimo +\_

reduce.

paralelno( i = 0; i < n; i++) {

UNIQUE[ i ] = 1;

K [ i ] = P [ i ];

}

~~for( i = 0; i < n; i++) {~~

~~paraleleno(j = 0; j < n; j++) {~~

~~if(P[ j ] == K [ j + i % n])~~

~~UNIQUE[ j ] = 0;~~

~~}~~

~~}~~ //Pogrešan kod je precrtan, a ne obrisan jer se dogodila rasprava dolje koja objašnjava zašto je pogrešan.

for ( i = 1; i < n; i++) {

paralelno( j = 0; j < n - i; j++) {

if( P [ j ] == K[ j + i ] )

UNIQUE[ j + i] = 0;

}

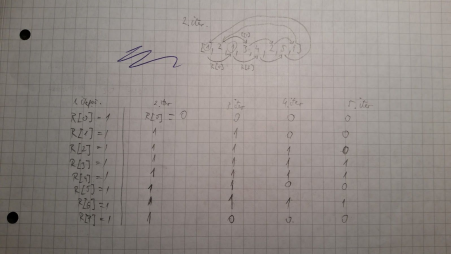
}

Rez = +\_reduce(UNIQUE []);

Ako netko ima rješenje za EREW koje troši manje memorije, molim vas napišite ga.

Da li bi trebalo zamijeniti vrijednosti UNIQUE[j] = 0 -> 1 i UNIQUE[i] = 1 -> 0? Kad se raspisu iteracije po ovom rjesenju, sa +reduceom se dobije broj elemenata koji se samo jednom pojavljuju u nizu, a ne broj razlicitih elemenata niza?

Možeš li staviti sliku raspisa? Ne vidim to u svojem raspisu, a lako je moguće da sam krivo raspisao.



Znaci ako su elementi isti, postavis ih na nulu i nikad vise ne poprime 1. Krivi mi je R[7] u 1. i 2.iteraciji, trebao bi biti 0.

U 2 iteraciji, stavljaš R[0] = 0. Zašto? Paralelno blok kreće od i+1 do n (u prvom slučaju je to od 1 do n), pa će se 0 stavit na index 2 i index 7. Index 0 ostaje nepromijenjen.

Gledamo EREW kod (ovaj iznad)? Tu paralelno pocne od j=0 zar ne?

Aha, za EREW si u pravu.

Može se uvjet promijenit u if(j+i < n && P[ j ] == K [ j + i])? Cini mi se da bi onda zadnji duplikat uvijek trebao ostati na 1.

Može ili tako, ili se može zaobići ~~ciklička~~ ~~usporedba~~ korištenje modulo n operacije. U smislu da se svaki element u polju P uspoređuje samo s elementima u polju K koji su strogo većeg indeksa od tog elementa u polju P. Na taj način se osigura da od tog elementa u polju P, ako je bilo duplikata desno od njega UNIQUE je promijenjen. Za duplikate lijevo od njega sam taj element je njima desno, pa će ga oni obraditi.

DJ: moj algoritam za EREW radi ovako: (ne gledaju se elementi "iza" zadnjeg, samo do zadnjeg u nizu)

P[] 1 2 1 3 4 2 5 1

U[] 1 1 1 1 1 1 1 1

d=1 1 1 1 1 1 1 1 1

d=2 0 1 1 1 1 1 1 1

d=3 0 1 1 1 1 1 1 1

d=4 0 0 1 1 1 1 1 1

d=5 0 0 0 1 1 1 1 1

d=6 0 0 0 1 1 1 1 1

d=7 0 0 0 1 1 1 1 1

na kraju +\_reduce

1.3 Napišite algoritam složenosti log(n) za EREW PRAM računalo koji će za zadano polje A[] s n elemenata odrediti je li uzlazno sortirano. Na raspolaganju je funkcija +\_reduce(polje[]) koja provodi operaciju zbrajanja.

paralelno( i = 0; i < n; i++) {

K [ i ] = A [ i ];

R [ i ] = 1;

} // O (1)

paralelno(i = 0; i < n - 1; i++) {

if( A [ i ] > K [ i + 1])

R [ i ] = 0;

} // O (1)

rezultat = +\_reduce(R[]); // O ( log n );https://www.youtube.com/watch?v=HneiEA1B8ks

if( rezultat == n)

//Polje je sortirano

složenost O(log n)

Ovo rješenje koristi dodatna dva polja. Mislim da se R[] može izbaciti te svaki procesor upisuje rješenje usporedbe u A[] na svojem indeksu. Također se zahtjeva da je metoda A.size() O(1).

Ne treba A.size() kad se zna da je n elemenata u nizu.

Hvala, ispravio.

1.4 Napisati algoritam za EREW PRAM računalo složenosti manje od n koji će za dva ulazna niza znakova A[] i B[] duljine n odrediti koji je prvi po abecedi (program treba ispisati "A", "B" ili "jednaki"). Na raspolaganju je funkcija reduciranja koja provodi proizvoljnu binarnu operaciju nad elementima polja. Ocijeniti složenost algoritma. Proizvoljnu operaciju možete definirati programski.

Rješenje zadatka je preuzeto iz videa PRAM zadaci (2). Stvori se funkcija nađi:

nađi(a,b) {

if( a == 0) {

return b;

}else {

return a;

}

}

paralelno ( i = 0; i < n; i++) {

if( A[ i ] < B [ i ]) {

R[ i ] = - 1;

}else if( A [ i ] > B [ i ]) {

R [ i ] = 1;

}else

R[ i ] = 0;

}

odg = nađi\_reduciraj( R [] );

if( odg == -1) {

odg = “A”;

}else if(odg == 1) {

odg = “B”;

}else {

odg = “jednak”;

}

složenost O(log n)

1.5 Napišite algoritam za EREW PRAM računalo složenosti manje od O(n^2) koji će za matricu M[n][n] odrediti predstavlja li permutacijsku matricu (u svakom retku i stupcu je točno jedna jedinica, a ostali elementi su nule). Program treba ispisati "da" ili "ne". Na raspolaganju je n procesora i funkcija reduciranja (O(log n)) koja provodi proizvoljnu binarnu operaciju nad elementima niza. Pretpostavite da su elementi matrice bilo koje cjelobrojne vrijednosti (mogu biti različiti od nula i jedan!), te da se i-ti redak odnosno stupac matrice može dobiti kao R[i] odnosno S[i]. Ocijenite složenost algoritma. Proizvoljnu operaciju možete definirati programski (u obliku funkcije).

složenost O (n log n)

// Profesore je li dobro ovo rješenje?

Definiramo asocijativnu binarnu operaciju ovako:

0 i 0 = 0;

0 i 1 ili 1 i 0 = 1

1 i X ili X i 1 (X != 0) = -1

0 i X ili X i 0 (X != 0 ili 1) = -1

-1 i X ili X i -1 (X e Z) = -1

ALGORITAM:

PARALELNO(i = 0 DO n -1){

REZ[2i] = op\_REDUCE(R[i]);

REZ[2i + 1] = op\_REDUCE(S[i]);

}

Suma = +\_REDUCE(REZ[]);

AKO JE(suma == 2n){

“DA”;

}INACE{

“NE”;

}

// Djeluje mi dobro, ali nisam siguran da nisam nešto previdio - Luka M

// Nisam siguran je li u redu koristiti reduce unutar paralelno bloka?

**//Jos jedno moguce rj?**

//Ideja:

//1.Pretvoriti sve brojeve u matrici u pozitivne ako su negativni

PARALELNO(i=0 DO n-1){

Za svaki element x iz R[i] { //Mijenjamo sve elemente tog retka, nisam znao kako to dobro oznacit ako(x<0) x\*=-1; //Naravno ovo je jedino moguce ako se smiju mijenjati elementi matrice

// Tu se također može program zaustaviti ukoliko je (x!=1 && x!=0) i ispisati “Ne” kao alternativa mijenjanja same matrice

}

Sum[i]=0;

}

//2.+\_reduce svaki stupac, +\_reduce svaki redak

PARALELNO(i=0 DO n-1){

Sum[i]+= +\_reduce(S[i]);

}

PARALELNO(i=0 DO n-1){

Sum[i]+= +\_reduce(R[i]);

}

//3.Ako je bilo koji redak/stupac != 2 => odgovor je ne, inače da

if(MAX\_reduce(Sum[]) != 2 && MIN\_reduce(Sum[])!=2){

ISPIŠI "Ne";

}

Else{

ISPIŠI "Da";

} - TĐ

**//Jel možemo riješiti i sljedećim postupkom**

PARALELNO(i=0 do N-1) {

suma[i]=0;

razlicito[i]=0

}

PARALELNO(i=0 do N-1) {

OD j=0 DO N-1 {

AKO JE R[i][j]==1

suma[i]+=1

INACE AKO JE R[i][j]!=0

RAZLICITO[i]=1

}

}

AKO JE OR\_REDUCE(RAZLICITO)!=0

Print NE

Izadi

PARALELNO(i=0 do N-1) {

OD j=0 DO N-1 {

AKO JE S[i][j]==1

suma[i]+=1

}

}

REZ=+\_REDUCE(suma[])

AKO JE REZ==2\*N{

Print DA

}

INACE{

Print Ne

}

Ideja meni izgleda ok, samo varijabla RAZLICITO nije dobra za EREW, više procesora može upisati/čitati to odjednom. I mislim da OR\_REDUCE ti vraća 0 ako su ti sve vrijednosti 0 inače vraća 1, tako da taj dio bi se trebalo malo prepravit (naravno ako sam ja dobro skužio OR\_REDUCE).

Hvala na primjedbi! Mislim da sam ispravio

Mislim da je sad +\_REDUCE problem, on bi trebao proć kroz polje i zbrojit sve vrijednosti, ako sam dobro razumio program ti u sum[i] stavljaš broj jedinica koji je i-ti procesor vidio. Uz “+\_REDUCE == 2” ti provjeravaš da li su svi procesori pronašli skupa točno dvije jedinice (tj. matrica bi imala samo jednu jedinicu).

Potrebno je dodati samo da je REDUCE jednak 2N, zametnulo se

**//Jel možemo riješiti i sljedećim postupkom ?**

ABS\_+ (a,b){

if(a == -1 || b == -1 || a == N || b == N)

Return -IntMax

Return |a| + |b|

}

PARALELNO(i=0 do N-1) {

Rx[i] = ABS\_+\_reduce(R[i])

}

PARALELNO(i=0 do N-1) {

Sx[i] = ABS\_+\_reduce(S[i])

}

ResR = +\_Reduce(Rx)

ResS = +\_Reduce(Sx)

maxR = MAX\_reduce(Rx)

if(ResR == N && ResS == N && maxR == 1)

Return “DA”

Return “NE”

Tu vidim dva moguća problema, prvi je da samo u jednom polju imaš “N”, tada će svi retci/stupci osim jednog imati Rx[i] = 0 a taj jedan bi imao Rx[i] = N. Uvjet je tu ispunjen. Drugi problem je ako imaš nule i negativne jedinice, u zadatku je zadano da u svakom stupcu i svakom retku mora biti točno jedna 1. U ovom kodu bi prošla i matrica koja u sebi ima negativne jedinice.

Hvala na primjedbi, mislim da ovaj uvjet sad rješava ta 2 slučaja

Mislim da logika ovog rješenja nije baš na mjestu, cilj je provjeriti da li svaki redak i svaki stupac sadrži točno jednu jedinicu. Uvjet “RessR==RessS” će uvijek biti točan jer u koncu on samo uspoređuje zbrojeve svih elemenata matrice, nije bitno da li zbrajaš prvo preko stupaca ili preko redaka. Moguće da sam ja krivo skužio kod ali mi izgleda kao da bi ova matrica prošla iako nije ispravna:

0 2 0

1 0 0

0 0 0

Vidim grešk,u…

// Ja sam smislio malo jednostavnije rješenje

PERM = 1

ZA (i=0; i<n; i++)

AKO (MIN\_reduce(R[i]) != 0 ILI MIN\_reduce(S[i]) != 0)

PERM = 0

AKO (MAX\_reduce(R[i]) != 1 ILI MAX\_reduce(S[i]) != 1)

PERM = 0

AKO (+\_reduce(R[i]) != 1 ILI +\_reduce(S[i]) != 1)

PERM = 0

AKO (PERM == 1)

ISPIŠI "Da"

INAČE

ISPIŠI "Ne"

// Čini mi se da je dovoljno provjeriti min, max i zbroj za svaki redak, a složenost je i dalje O(n\*log(n))

^Mislim da u EREW PRAM Ne mozes imati PERM varijablu

To je varijabla kojoj pristupa samo 1 procesor (ajmo reć master), ne vidim zašto ne //SLAŽEM SE,ni meni nije jasno zašto ne

//eventualno problem može biti što nema bloka paralelno?

// ¯\\_(ツ)\_/¯ // pa sta nije mozda ideja da se napisu paralelni algoritmi a ne da sekvencijalno rijesavamo? Cemu onda razlikovanje u zadatku EREW ili CRCW ako ces rijesitit s jednim “procesorom” ? // Ovo nije sekvencijalno rješavanje, funkcije x\_reduce su paralelne po svojoj prirodi. Ovaj algoritam se može izvršavat na bilo kojem računalu (EREW/CREW) s nlogn složenošću, ne znači da je loš jer koristi jedan “glavni” procesor.

1.6 Napisati algoritam za CRCW PRAM računalo koji će, ispisivanjem "DA" ili "NE", za zadano polje P[] s n elemenata odrediti predstavlja li permutacijski vektor (permutacija skupa {1, 2, ..., n}). Elementi polja mogu poprimiti samo cjelobrojne vrijednosti. Primjerice, (3, 1, 2) je permutacijski vektor dok (2, 4, 3), (1, 1, 2) i (4, 2, 1) nisu. Na raspolaganju je funkcija reduciranja koja provodi proizvoljnu binarnu operaciju nad elementima polja. Ocijeniti složenost algoritma.

//postavljanje globalne varijable

PERMUTACIJE = TRUE

//postavljanje pomoćnog polja u kojeg pišemo rezultat

PARALELNO (i = 0 do n-1)

POM[ i ] = 0

//svakom procesoru dodijelimo jedan element polja i on će uzeti vrijednost tog elementa i u pomoćno polje na mjesto te vrijednosti (umanjeno za 1 jer indeksi kreću od 0) upisati vrijednost 1 PARALELNO (i = 0 do n-1)

//ispitamo jel vrijednost 0, ako je upišemo da smo taj element našli u početnom polju AKO (POM [ P [i] -1] ==0)

POM[ P[i]-1] = 1

//inače znači da je vrijednost 1 što znači da smo taj element već vidjeli u početnom polju i znamo da nije permutacija jer ne smije biti duplikata

INAČE

PERMUTACIJE = FALSE

AKO (PERMUTACIJE == TRUE)

//radimo AND\_REDUCE nad poljem koje bi trebalo na svakom elementu imati vrijednost 1 (znači da smo vidjeli svaki element polja)

REZ = AND\_REDUCE( POM[])

AKO (REZ==0)@

“NE”

INAČE

“DA”

INAČE

“NE”

Primjer izvršavanja

P = [1 3 2 4]

Proces koji je dobio indeks (i) 0 će uzeti vrijednost P[0] (1) te u pomoćno polje upisati vrijednost 1 na mjesto P[0] -1 = 1 - 1 = 0

POM = [ 1 0 0 0]

Proces koji je dobio indeks(i) 1 će uzeti vrijednost P[1] (3) te u pomoćno polje upisati vrijednost 1 na mjesto P[1]-1 = 3 - 1 = 2

P = [ 1 0 1 0]

…

P = [1 1 1 0]

….

P = [1 1 1 1 ]

AND\_REDUCE vrati 1 i rješenje je “DA”

P = [2 3 3 1]

POM=[ 0 1 0 0]

…

POM=[0 1 1 0]

….

Sad bi opet trebali upisati vrijednost 1 na mjesto POM[2], ali pošto tamo već je upisana vrijednost 1 znači da smo taj element već vidjeli i postavljamo globalnu varijablu PERMUTACIJE=FALSE

Složenost ovog algoritma bi bilo O(log n)

DJ: OK, napomene:

POM [ P [i] -1] - ovo se NE BI SMJELO raditi, jer ne znamo koje vrijednosti mogu poprimiti elementi niza P[]; dakle ne koristiti neizravno adresiranje!

jednostavnije (?) rjesenje ovog problema: prvo provjeriti jesu li svi elementi niza razliciti, odnosno da nema duplikata (taj primjer smo rjesavali); onda samo napravimo MIN-reduce i MAX-reduce, i ako su min i max vrijednosti 1 i n, to je permutacija...

Može li se POM [ P [i] -1] koristiti ako prije toga stavimo provjeru je li broj u polju između 1 i n?

DJ: da, onda moze. Ne svidja mi se, al moze.

Je li ovakvo rjesenje prihvatljivo? Slozenost je O(log n).

rez = 0;

err = 0; // nema greske zasad

paralelno (za i = 0 do n-1)

ako (P[i] > n ili P[i] < 1) // vrijednost vektora ne smije izlaziti van intervala [1, n] err = 1;

ako (err = 0) // nema elementa van intervala [1, n]

rez = +\_reduce(P[]);

ako (rez == (n\*(n+1))/2) // zbroj vrijednosti mora biti jednak (n\*(n+1))/2 ispisi "DA";

inace

ispisi "NE";

inace

ispisi "NE";

složenost O(n)

Mislim da ovo ne valja jer suma prvih n brojeva se može dobiti i uz drukčije brojeve (npr. 1234 i 1333)

DJ: tocno, nije dovoljno provjeriti samo zbroj i [min, max]

1.7 Napisati algoritam za EREW PRAM računalo koji će za zadano polje P[] s n elemenata odrediti predstavlja li neki redak permutacijske matrice (jedan element 1, svi ostali 0). O vrijednostima elemenata polja se ništa ne pretpostavlja (mogu biti bilo koje vrijednosti). Na raspolaganju je funkcija \*\_reduce(polje[]) koja provodi proizvoljnu binarnu operaciju nad elementima polja. Ocijeniti složenost algoritma.

složenost O(log n)

NULE = [];

JEDINICE = [];

PARALELNO (i = 0 DO n-1)

AKO (P[I] == 0)

NULE[i] = 1;

INAČE

NULE[i] = 0;

AKO (P[i] == 1)

JEDINICE[i] = 1;

INAČE

JEDINICE[i] = 0;

R0 = +\_REDUCE(NULE[]);

R1 = +\_REDUCE(JEDINICE[]);

AKO (R0 == n-1 && R1 == 1)

ISPIŠI “DA”;

INAČE

ISPIŠI “NE”;

1.8 U ostvarenju igre 4 u nizu prazni elementi ploče su označeni nulom, a pozicije s igračevim žetonom jedinicom (ne postoje elementi drugih vrijednosti). Napišite algoritam za EREW PRAM računalo koji za zadani (jednodimenzijski) niz elemenata ploče P[] duljine n otkriva postoji li u njemu 4 igračeva žetona u nizu (ispisuje DA ili NE). Na raspolaganju su scan i reduce funkcije za proizvoljne operacije. Netrivijalne operacije (npr. one koje uključuju grananja) potrebno je definirati algoritamski.

složenost O(log n)

U videu je navedeno kako je X\_scan funkcija kriva jer nije asocijativna. Ako definiramo X na sljedeci nacin:

X(a,b):

Ako b == 0:

Vrati a // ovdje se u videu vraćala 0

Inače:

Vrati a+b

Tada je bi X\_scan bio asocijativan i mogao bi se jednostavno riješiti zadatak.??

R[] = X\_scan(P[])

Max = MAX\_reduce(R[])

Ako Max >= 4: “DA”

Inace: “NE”

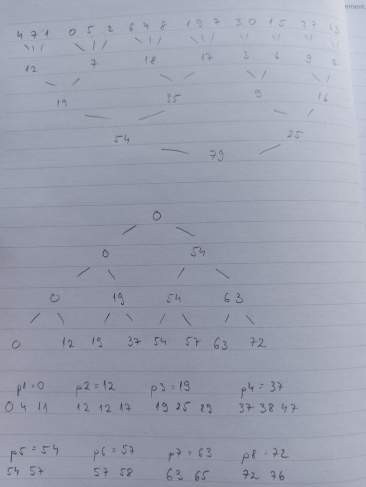
Mislim da ti ovo nije dobro, jer za niz “1 0 1” ti dobivaš uvijek 2. Što nije točno, najduži niz jedinica bi trebao uvijek vraćati 1 za taj slučaj. Edit: Neka ostane, možda sam i ja nešto fulao

Istina da, možeš izbrisati sve da ne bunimo ostale. Tnx

------------------

Može li se provesti npr. 4 prolaza kroz polje. U prvom prolazu kopiramo polje u novo. U svakom idućem gledamo prvo jedno desno, pa 2 desno, pa 3 desno. Kad naiđemo na 0 na mjestu koje gledamo, u početnom mjestu stavimo 0, inače ako je tamo 1, ne diramo. Na kraju imamo 1 na mjestima na kojima počinje rješenje od 4 u nizu. Kasnije provedemo max reduce ili sum i vidimo dobijemo li > 0. To je log(n), samo zbog tog reduce na kraju.

1.9 Provesti *+\_prescan* algoritam na zadanom polju duljine *n* = 20 elemenata i na *p* = 8 procesora. Označiti podjelu elemenata po procesorima i tablično napisati izvedbu algoritma. Ulazno polje je A[] = [4 7 1 0 5 2 6 4 8 1 9 7 3 0 1 5 2 7 4 3].



1.10 Napišite algoritam za EREW PRAM računalo koji za zadani niz cjelobrojnih vrijednosti duljine n uz najviše n procesora otkriva duljinu najduljeg neprekinutog podniza elemenata s jednakom vrijednošću (ispisati duljinu podniza). Na raspolaganju su scan i reduce funkcije za proizvoljne operacije. Netrivijalne operacije (npr. one koje

uključuju grananja) potrebno je definirati algoritamski. Ocijenite složenost algoritma. složenost O(log n)

***Ima li itko postupak za ovo? Hvala :)***

Paralelno i = 0 do n-1

K[i] = P[i]

R[i] = 1// zasto ovo ide u 1? //zato sta se svaki element pojavi bar jednom ako postoji // kaj u slucaju ovakvog niza 1,2,2,2,3,4? Ti postavis R od 0 do 5 sve na 1 // mozda krivi niz haha kaj ako je 3,3,3,1,5?

//ne kuzim zasto je to problem napisat cu dolje primjer

Za j=1 do n-1

Paralelno(i=0 do n-2)

Ako(R[i] == j)

Ako(P[i] == K[i+j] // mislim da ne mozes ovo raditi ako je EREW

//usporeduju se dva niza i s obzirom da se paralelno

izvrsava **i** a **j** je fiksiran nebi trebalo bit problema

R[i] ++;

M = MAX\_Reduce(R[])

Složenost : O(n)

**Može netko ovo provjeriti?**

P[] = 3 3 3 1 5

R[] = 1 1 1 1 1

J = 1 2 2 1 1 1

J = 2 3 2 1 1 1

Na kraju se uzme max to je 3 i to je to.

Neće li ako imamo P[] = 3 3 3 1 5 3 3

Rezultat biti 5 što je krivo

Neće jer imas uvjet R[i] = j

Mislim da je uvjet u “paralelno (i= 0 do n-2)” krivi, da bi trebalo ici do (n-j-1) jer iskace van polja?

Probao sam sad sa PRAM simulatorom ovo, za CRCW radi, no za EREW izbacuje grešku o memory reading constrain violation

Mozda jer svaki proces cita istu varijablu ( j ) pa onda baca violation? (Yes)

Ja sam to ovako:

Paralelno i = 0 do n -1

K[i] = P[i]

R[i] = 0

Za j = 0 do n - 1

Index = K[i]

R[index] = 1 // npr imamo niz 1 2 2 2 3 4 5, j=0, index = K[0] = 1 R[1] = 1 Paralelno (z = j +1 do n -1)

Ako index == K[z]

R[index]++

Ali mi se sad to ne cini dobro jer bi mozda moglo duplo brojati

Mozda bi samo trebalo pogledati sljedeceg i ako su isti onda zbrojiti

Možda:

paralelno (i=0 do n-1)

K[i] = P[i];

flag[i] = 1;

zbroj[i] = 1;

paralelno (i=0 do n-1)

za (j=1 do n-1-i)

ako (K[i+j] != P[i])

flag[i] = 0;

ako (flag[i] == 1)

zbroj[i] += 1;

rez = max\_reduce(zbroj[]);

?

?Jel moguće ovo sa scan ?

**2 MPI**

2.1 Korištenjem MPI funkcija Send i Recv (skraćena sintaksa) napišite niz instrukcija koji će sve elemente zadane kružne liste postaviti na srednju vrijednost toga i dvaju susjednih elemenata (indeksi i, i+1, i-1; posljednji element povezan je s prvim i obrnuto). Svaki MPI proces ima u lokalnoj memoriji samo jedan element liste koji je realna vrijednost. Broj procesa je N, svaki proces ima redni broj ID. *Program treba jamčiti ispravnost rada bez obzira na veličinu poruka* (ne smije doći do potpunog zastoja zbog redoslijeda slanja i primanja)!

Rješenje: Svaki proces treba izvršiti iduće:

● Poslati svoju vrijednost lijevom susjedu

● Poslati svoju vrijednost desnom susjedu

● Primi vrijednost lijevog susjeda

● Primi vrijednost desnog susjeda

● Izračunaj srednju vrijednost

Osim što svaki proces u svojoj lokalnoj memoriji ima svoj element liste, ID i N, također treba imati jednu varijablu (buffer) u koju će primati vrijednosti od susjeda. Ta varijabla se dodaje vlastitom elementu liste te se u konačnici taj element dijeli sa 3 (srednja vrijednost).

Isječak koda (C++):

// …..

double my\_val\_copy = my\_val;

MPI\_Send(&my\_val,1,MPI\_DOUBLE,(m\_rank+w\_size-1)%w\_size,0,MPI\_COMM\_WORLD);

MPI\_Recv(&m\_buff,1,MPI\_DOUBLE,(m\_rank+w\_size-1)%w\_size,0,MPI\_COMM\_WORLD,&l s);

my\_va\_copyl += m\_buff;

MPI\_Send(&my\_val,1,MPI\_DOUBLE,(m\_rank+1)%w\_size,0,MPI\_COMM\_WORLD); MPI\_Recv(&m\_buff,1,MPI\_DOUBLE,(m\_rank+1)%w\_size,0,MPI\_COMM\_WORLD,&ls); my\_val\_copy += m\_buff;

my\_val\_copy /= 3.0;

Author’s note: Ako netko zna ili vidi kako bi ovaj program mogao izazvati potpuni zastoj ili neke ostale probleme neka napiše.

Ovo je više pitanje, jer nisam sigurna, ali zar neće ovo uzrokovati potpuni zastoj u slučaju da funkcija SEND radi u synchronous modeu, dakle ako čeka da primatelj pozove RECV, onda će svi procesi zapeti nakon prve SEND naredbe?

U pravu si, napravio sam reordering da ide SEND,RECV,SEND,RECV

DJ 28.6.: potrebno je osigurati da barem jedan proces na pocetku radi RECV; ne smiju *svi* procesi imati SEND na pocetku, to je vec dovoljno za zastoj (u teoriji, zbog blokirajuce komunikacije)

*Kako da to osiguramo?*

:) ima vise nacina: recimo neparni prvo salju, parni prvo primaju? uglavnom treba ih podijeliti na barem dvije razlicite uloge

IF ID % 2 == 0 // parni

Send (moj\_podatak, ID+1)

Send(moj\_podatak, ID-1)

Recv (pod\_desno, ID+1)

Recv (pod\_lijevo, ID-1)

IF ID % 2 == 1 // neparni

Recv (pod\_lijevo, ID-1)

Recv (pod\_desno, ID+1)

Send (moj\_podatak, ID-1)

Send (moj\_podatak, ID+1)

moj\_podatak = (pod\_lijevo + moj\_podatak + pod\_desno) / 3

Nek netko javi je li ovo okej :)

Samo stavis da ti npr proces 0 prvo recv pa send --- tada ce se kruzno izvrsavati send pa recv naredbe i uvijek ce jedan proces imati uspješan send pa aktivirani recv

Znači ovo gore se treba prepraviti tako da samo za id=0 radi prvo recv ili može i ovako?

2.2 U jednom trenutku rada paralelnog programa u *n* procesora se nalaze neki podaci. Potrebno je odrediti najveći element od svih *n* podataka i tu informaciju (vrijednost najvećega) proslijediti svim procesorima. Napisati algoritam koji će obaviti taj zadatak pomoću MPI funkcija MPI\_Send i MPI\_Recv. Uputa: slanje i primanje poruka obaviti u obliku lanca u dva prolaza (s lijeva na desno te potom s desna na lijevo po svim procesorima). Kod poziva MPI funkcija navesti samo 'bitne' parametre (npr.

MPI\_Send(&varijabla,\_,\_,odrediste,\_,\_); ).

If (my\_rank == 0)

MPI\_Send(&my\_number,\_,\_,1,\_,\_); // proces 0 šalje procesu 1

else if(my\_rank > 0 && my\_rank < world\_size - 1) {

MPI\_Recv(&rec\_buff,\_,\_,my\_rank-1,\_,\_,\_); // prvo primi od procesa lijevo od sebe my\_number = rec\_buff >= my\_number ? rec\_buff : my\_number;

MPI\_Send(&my\_number,\_,\_,my\_rank+1,\_,\_); //šalji jednom desno od sebe. }

else

MPI\_Recv(&rec\_buff,\_,\_,my\_rank-1,\_,\_,\_);

my\_number = rec\_buff >= my\_number ? rec\_buff : my\_number;

if(my\_rank == world\_size - 1)

MPI\_Send(&my\_number,\_,\_,my\_rank-1,\_,\_);

else if(my\_rank > 0 && my\_rank < world\_size - 1) {

MPI\_Recv(&rec\_buff,\_,\_,my\_rank+1,\_,\_,\_); // Prvo primi od procesa desno od sebe. my\_number = rec\_buff >= my\_number ? rec\_buff : my\_number;

MPI\_Send(&my\_number,\_,\_,my\_rank-1,\_,\_);

}

else {

MPI\_Recv(&rec\_buff,\_,\_,my\_rank+1,\_,\_,\_);

my\_number = rec\_buff >= my\_number ? rec\_buff : my\_number;

}

DJ: ovo je dobro, samo nije potrebna puna sintaksa vec samo SEND/RECV(<podatak>, <PID>)

Je li potrebno u drugom prolazu provjeravati (s desna na lijevo) je li primljeni podatak veci? Zar se nece otkriti najveci broj u prvom prolazu pa jednostavno samo uzeti taj primljeni podatak ko rjesenje?

DJ: tocno, u povratku se ne bi trebalo nista provjeravati

2.3 Korištenjem MPI funkcija *Send* i *Recv* (skraćena sintaksa) napisati odsječak programa (proizvoljne složenosti) koji će za *N* procesa ostvariti funkciju MPI\_Barrier, tj. postići da svi procesi moraju doći do istog odsječka prije nego bilo koji proces može nastaviti s izvođenjem. (U svakom procesu varijabla *ID* je indeks, a varijabla *N* ukupni broj procesa.)

1. Kada se dođe do trenutka gdje se želi ostvariti barijera potrebno je svim ostalim procesima poslati poruku da se došlo do barijere. (n - 1 poruka)

2. Svaki proces treba primiti (n -1 poruku) od drugih procesa da su stigli do barijere. To se ponavlja n puta, pa se ukupno prima n \* ( n - 1 ) poruka.

Sveukupno je ovo ( n + 1 ) \* ( n - 1 ) poruka. Otprilike n^2 poruka.

DJ: moguce, ali nije potrebno da svaki javi svakome; mogu svi javiti *jednome* pa taj jedan svima javi povratnu informaciju (tek kad dobije poruke od svih), ukupno 2(n-1) poruka

I naravno moguce je uz hiperkocku, ali nije potrebno ako nije navedena slozenost. Je li ovo u redu rješenje?

If (my\_rank == 0) {

br\_javljenih = 0;

While (br\_javljenih < N-1) {

RECV(&poruka, \*)

If (poruka == “stigao”) {

Br\_javljenih += 1;

}

}

For (i = 1 do N-1) {

SEND(“barijera”, i);

}

} else {

SEND(“stigao”, 0);

Poruka = “”;

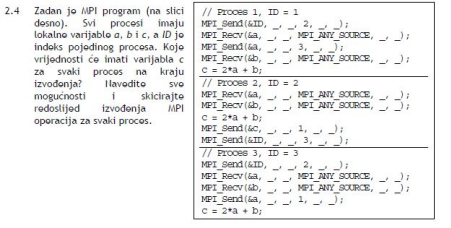
while(poruka != “barijera”) {

RECV(&poruka, 0);

}

}

DJ: je, samo za ostale procese nije potrebna while petlja, njima je dovoljan send pa recv. U praksi bi to riješili posebnim tagom ili komunikatorom (u zadacima nije potrebno).



Ovdje 4 mogućnosti ? Send procesa 1 dode prvi do procesa 2 ili Send procesa 3 dode prvi do procesa 2. I za svaki od ta dva slucaja ili Send broj 2 u procesu 1 dode prvi do procesa 3 ili Send broj 2 procesa 2 dode prvi do procesa 3.

DJ: da, 4 mogucnosti. Konkretne vrijednosti varijabli su manje bitne.

2.5 U MPI programu svaki proces ima lokalnu vrijednost u varijabli *x*. Korištenjem MPI funkcija *Send* i *Recv* (skraćena sintaksa) napisati odsječak programa logaritamske složenosti (po pitanju broja poslanih poruka) koji će za *N* procesa izračunati minimum svih lokalnih vrijednosti, tako da svi procesi znaju rezultat. (U svakom procesu varijabla *ID* je indeks, a varijabla *N* ukupni broj procesa.)

Koristi se struktura hiperkocke

ZA (i = 0; i < logN; ++i) {

DEST\_ID = ID XOR 2^i;

SEND(x, DEST\_ID);

RECV(njegov\_x, DEST\_ID);

If (njegov\_x < x) {

x = njegov\_x;

}

}

Moze li se ovaj a i sljedeca dva zadatka rjesiti putem principa binarnog stabla? Ukoliko broj elemenata nije potencija broja dva, prilagodimo elemente na nacin da ih slijedno pridruzimo

zadnjem elementu na potenciji broja dva (primjerice ako imamo 9 elemenata, odradimo slijedno operaciju na 8. I 9.) te onda provesti SEND i RECV?

DJ: da, a u zadacima nije potrebno gledati konkretni broj procesa (uzimamo da jest potencija broja 2)

2.6 U MPI programu u nekom trenutku pojavio se promatrani događaj unutar jednog MPI procesa. Svi procesi znaju da se događaj pojavio, ali nijedan proces (osim dotičnog, izvorišnog procesa) ne zna unutar kojeg procesa se pojavio događaj (tj. koji je proces izvorišni). Korištenjem MPI funkcija *Send* i *Recv* (skraćena sintaksa) napisati odsječak programa logaritamske složenosti (po pitanju broja poslanih poruka) koji će za *N* procesa omogućiti da svi procesi saznaju indeks izvorišnog procesa. (U svakom procesu varijabla *ID* je indeks, a varijabla *N* ukupni broj procesa)

Je li ovo dobro rješenje?

Izvor = -1;

If (događaj) izvor = ID;

ZA (i = 0; i < logN; ++i) {

DEST\_ID = ID XOR 2^i;

SEND(izvor, DEST\_ID);

RECV(njegov\_izvor, DEST\_ID);

If (njegov\_izvor > -1) {

izvor = njegov\_izvor;

}

}

DJ: yup

Kako ovo ne blokira? Jer svaki prvo radi Send? Treba li opet uvesti uloge ili? Mislim da ako ne pise da ne smije doci do potpunog zastoja nista ne radimo po tom pitanju. Zanimljivo.

DJ: u slucaju MPI-ja to se lako rijesi, recimo da podijelimo uloge na one s parnim i neparnim indeksima. Ali ovdje je rijec o univerzalnom algoritmu, koji se ne mora izvoditi samo s MPI, pa nisam htio zagadjivati originalni algoritam hiperkocke s "popravcima" za MPI.

U zakljucku: u zadacima ne morate brinuti o tome - jedino ako se eksplicitno trazi.

2.7 U MPI programu u nekom trenutku svih N procesa treba obaviti kritični odsječak. Svaki proces zna svoj redni broj ulaska u K.O., ali ne zna redne brojeve ostalih procesa. Korištenjem MPI funkcija Send i Recv (skraćena sintaksa) napisati odsječak programa logaritamske složenosti (po pitanju broja poslanih poruka) koji će omogućiti da svaki proces sazna indeks svog neposrednog prethodnika i sljedbenika (pozivanje K.O. nije potrebno

prikazati). U svakom procesu varijabla ID je indeks procesa, a varijabla RBR redni broj ulaska u K.O.

left,right=-1,-1

for(i=0;i<log(N);i++) {

dest=ID XOR 2\*\*i;

SEND(RBR,dest);

RECV(value,dest);

if(value==RBR-1 || value=N-1 && RBR=0) {

left=dest;

} else if(value==RBR+1 || RBR==N-1 && value=0) {

right=dest;

}

}

Kolko sam shvatio iz zadatka, svaki proces mora znati svog sljedbenika i prethodnika, a to nigdje u ovom kodu nije napravljeno, a i nisu mi jasni ovi uvjeti u if-u. U prvom ifu ako jer RBR = 0 zar ne bi trebali provjeravati onda dal je value == N-1, a ne value == RBR - 1 (jer ako RBR mora biti 0 i value mora biti RBR - 1, to bi znacilo da je value == -1 kaj nema smisla?), a u drugom ifu zar nebi trebalo provjeravat dal je onda value == 0, a ne RBR + 1, ako je nas RBR == n-1? Vodio sam se pretpostavkom da RBR moze biti od 0 do N-1.

Što se tiče if uvjeta, treba samo pisati da je različito od nula odnosno N-1, što sam sada izmijenio. Ja sam shvatio da treba zapamtiti ID procesa koji imaju susjedne RBR. EDIT: Mislim da sam skužio što si htio još reći, tako da sam izmjenioop

Moja ideja: Hiperkockom iz svakog procesa saljemo dictionary [ID:RBR]. Primanjem tog dictionarya svaki proces updatea svoju vlastitu mapu. Na kraju izvedbe algoritma svaki proces ima potpunu mapu sa svim ID-evima i pripadajucim rednim brojevima i onda iz te strukture gleda koji je sljedeci i koji je prethodni element.

Tom sam logikom i ja išao. Samo što sam spremao samo prvi manji odnosno veći (u left i right), jer nam drugi ne trebaju.

"saljemo dictionary" - joj… ne saljite dictionary, saljite niz fiksne velicine. Nije python los, ali ovo je uzelo maha…

**------------------------------- 2.7. ------------------------------- lista = int[N]**

**za(i = 0 do n-1):**

**lista[i] = -1**

**lista[RBR-1] = ID**

**za(i = 0 do log n):**

**dest\_ID = ID XOR 2^i**

**send(lista, dest\_ID)**

**recv(druga\_lista, dest\_ID)**

**za(i = 0 do n-1):**

**ako(druga\_lista[i] != -1 && lista[i] == -1):**

**lista[i] = druga\_lista[i]**

**prethodnik = RBR != 1 ? lista[RBR-1-1] : -1**

**sljedbenik = RBR != N ? lista[RBR-1+1] : -1**

DJ: tako nekako. Inace je \*moguce\* riješiti i bez slanja cijele liste, ali zaboravio sam kako ide to rjesenje ;)

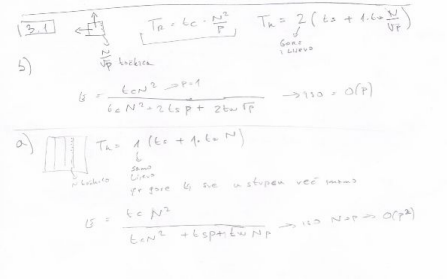
**3 Analiza**

3.1 Paralelni algoritam iterativno računa elemente matrice. Nova vrijednost elementa računa se pomoću vrijednosti neposrednih elemenata gore i lijevo, s tim da matrica ima 'spojene' sve vanjske bridove (npr. vrijednost elementa A[1,1] računa se pomoću A[N,1] i A[1,N]). Pretpostavite da trenutno stanje matrice omogućuje istovremeno računanje novih vrijednosti u jednoj iteraciji za sve elemente. Trošak računanja jednog elementa iznosi *tc*. Izrazite trajanje izvođenja jedne iteracije na P procesora te učinkovitost i izoučinkovitost algoritma ako je matrica na procesore podijeljena

a) po stupcima (svaki procesor ima jednak broj stupaca),

b) po podmatricama jednake veličine.

izoučinkovitost a) O(P^2), b) O(P)

Kopirano s materijala\*

Pod b) u nazivniku treba biti …2twN\*(p)^½, inače je dobro

Zašto se u a) dijelu zadatka u izrazu komunikacije uzima N kao duljina poruke ?

Zato jer su ti potrebne prilikom izračuna samo lijeve točke(kojih ima N, da su trebale dvije lijeve, veličina poruke bi bila 2N). Gornje točke već imamo jer radimo sa stupcima

Kako se dobije izoučinkovitost ?

3.2 Paralelni algoritam iterativno računa elemente kvadratne matrice veličine n x n. Nova vrijednost svakog elementa računa se pomoću vrijednosti svih elemenata u istom retku i istom stupcu. Pretpostavite da stanje matrice omogućuje istovremeno računanje novih vrijednosti u jednoj iteraciji za sve elemente. Trošak računanja jednog elementa iznosi tc. Izrazite trajanje izvođenja jedne iteracije na P procesora te učinkovitost algoritma ako je matrica na procesore podijeljena:

a) po stupcima (svaki procesor ima jednak broj stupaca):

b) po kvadratnim podmatricama jednake veličine:

Profesore, mozete li provjeriti postupak? Sto ako nam ostane p u izrazu kod izoucinkovitosti?

//mislim da je važno da bude približno ostvaren uvjet , E-->konst

DJ: u ovom se zadatku namjerno ne trazi izoučinkovitost; mora se uzeti jos veca potencija uz P kako bi nakon dijeljenja P ostao samo u nazivniku (samo s negativnim potencijama, kao primjer 1D atm modela u skripti), pa se onda moze priblizno uzeti. ali ovdje ispada ruzno (dobijemo i P i P^2 u nazivniku), sto je znak da se funkcija izouc. ne moze "lijepo" (jednostavno) izraziti. U zadacima ce uvijek biti moguce svesti na oblik kakav smo radili u zadacima.

//ali koliko god velik p uzmemo neće biti dobro jer će uvijek uz tw biti p\* više(a npr), što bi značilo da je ovo rješenje ok il?

trebali bi uzeti vecu potenciju n za N = P^n, onda se nakon dijeljenja dobiju P s negativnim potencijama. ali to samo pokazuje da je izouc zapravo puno veca od recimo kvadratneje

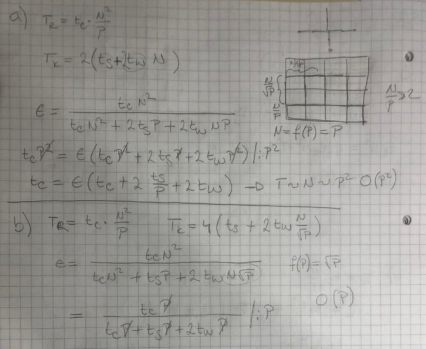
Po meni je veličina poruke samo N/p jer treba slat samo jedan redak koji je dugačak N/p. Može netko to potvrdit?

DJ: pa ne, i redak i stupac su velicine N, a u slucaju a) trebaju se slati svi podaci jednog zadatka

3.3 Paralelni algoritam iterativno računa elemente matrice. Nova vrijednost elementa računa se pomoću vrijednosti 2 neposredna susjedna elementa u svakom od 4 smjera, s tim da matrica ima 'spojene' sve bridove (npr. vrijednost elementa A[1,1] računa se pomoću A[1,2], A[1,3], A[1,N-1], A[1,N], A[N-1,1], A[N,1], A[2,1] i A[3,1]). Trošak računanja jednog elementa iznosi *tc*. Izrazite trajanje izvođenja jedne iteracije na P procesora te učinkovitost i izoučinkovitost algoritma ako je matrica na procesore podijeljena:

a) po retcima (svaki procesor ima jednak broj redaka),

b) po podmatricama jednake veličine.

3.4 Paralelno računalo plaća se 2 kn po satu po procesoru

. Trajanje izvođenja slijednog programa je dva tjedna. Učinkovitost paralelnog programa koji obavlja isti posao dan je izrazom 

a) Koliko ćemo platiti rad paralelnog računala ako program želimo obaviti za 5 dana? 6720 kn? DJ: tocno

b) Na koliko procesora se program mora izvoditi? 28? DJ: tocno

c) Koje je minimalno trajanje paralelnog programa (uz proizvoljni broj procesora)? 14/3 dana? -> 112h Moze netko napisati kako se do ovoga dode DJ: tocno

T1 = 14 dana

Ep = 3 / (2 + p)

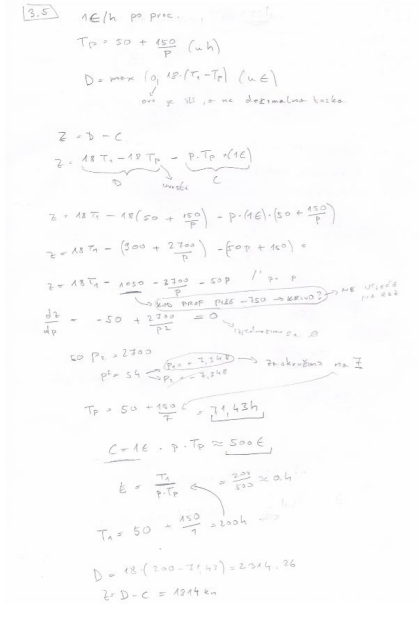
Ep = T1 / (P \* Tp) -> Tp = T1 / (Ep \* P) = 14 / (3 P / (2 + P))

Riješiš dvojni razlomak

Ep = (28 + 14P) / 3P = 28/3P + 14P/3P = 14/3 + 28/3P

Za dovoljno veliki P član 28/3P teži u 0 pa ti ostane 14/3 što je minimalno vrijeme

3.5 Paralelno računalo plaća se 1 kn po satu po procesoru. Na raspolaganju nam je paralelni program čije se trajanje izvođenja može izraziti kao (u satima). Dobit od rezultata izvođenja programa ovisi o trenutku dobivanja rezultata i opisana je izrazom (u kunama). Koje trajanje izvođenja nam donosi najveću moguću zaradu (dobit – troškovi) i na koliko procesora?

DJ: jest!

//////Da je rezultat bio 7.5 jel bi onda zaokružili na 8 procesora???

DJ: matematicki gledano, da; u ispitu bi priznali obje varijante, ali zato u ispitu nece biti .5; no ako je recimo 7.6 onda ide na 8

P = 7

3.6 Paralelni program računa nove vrijednosti matrice veličine *n* x *n* tako da transponira matricu, a zatim računa novu vrijednost svakog elementa (računanje vrijednosti jednog elementa traje *tc*). Izrazite trajanje izvođenja ove operacije na P procesora te učinkovitost i izoučinkovitost algoritma ako je matrica na procesore podijeljena:

a) po stupcima (svaki procesor ima jednak broj stupaca)

b) po podmatricama jednake veličine.

Anybody ?

https://www.youtube.com/watch?v=MPE-8JU3bvU&feature=youtu.be

izoučinkovitost a) O(P^2), b) O(P)

3.7 Paralelni program kvadrira matricu *n* x *n* uz primjenu standardnog algoritma za množenje matrica (za jedan element umnoška, zbrajaju se umnošci parova elemenata u odgovarajućem retku i stupcu). Matrica je na podijeljena na P procesora po podmatricama

iste veličine. Složenost jedne operacije množenja je *tc*(zbrajanje se zanemaruje). Odredite trajanje operacije kvadriranja te izoučinkovitost algoritma.

izoučinkovitost O(P^(3/2))