Paralelno programiranje

Završni ispit

ak. god. 2009./2010.

1. (6 bodova)-

U ostvarenju igre četiri u nizu prazni elementi ploče označeni su nulom, a pozicija 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 četiri igračeva žetona u nizu (ispisati DA ili NE). Na raspolaganju su *scan* i *reduce* funkcije za proizvoljne operacije. Netrivijalne operacije (npr. one koje ne uključuju grananja) potrebno je definirati algoritamski.

```
FOR i=0 do 3 {
     PARALELNO (j = 0 DO n/4) {
          //provjeri p[j*4 + i] do p[j*4+i+3] da li su četiri //
          jedinice u nizu
          ako jesu {
               r[j]=1;
          } inace {
               r[j]=0;
          }
      rez = OR_REDUCE(r[]);
      AKO (rez == 1)
           ispisi(“DA”);
      INAČE
           nastavi s FOR petljom;
      } // kraj PARALELNO bloka
} // gotova je FOR petlja
ispiši(“NE”)
```

Složenost O(logN)

Yowza: moglo bi se komotno pisat rješenja jedna ispod drugih...

Rješenje by Bambi®:

```
PARALELNO (i = 0 DO n - 1)
      q[i] = p[i]; // zbog ER
      r[i] = 0;    // zbog EW
PARALELNO (i = 0 DO n - 4)
      rez = 1;
      AKO (p[i] == 1)
            ZA (j = 1 DO 3)
                  AKO (p[i] != q[i + j])
                        rez = 0;
      INAČE
            rez = 0;
      AKO (rez == 1)
            r[i] = 1;
rez = OR_REDUCE(r[]);
AKO (rez == 1)
      ispisi(“DA”);
INAČE
      ispisi(“NE”);
// složenost je logaritamska
```

Rjesenje by Shefinho:

```
PARALELNO ZA i=0 DO N-1
    r[i] = 0; // zbog EW

PARALELNO ZA i=0 DO N-4
    r[i] = 1;
    ZA j = 0 DO 3
        AKO (p[i+j]!=1) // ER inherentno osiguran :-)
            r[i] = 0;

rez = OR_REDUCE(r[])
AKO (rez == 1)
    ispiši("DA");
INAČE
```

```
    ispiši("NE");
```

Ovo “inherentno osiguran” znači da je svaki proces dobio kopiju od p[]?

Ovo rješenje je krivo!

Dobro je rješenje jer se paralelno sve izvršava. Tako da paralelno procesori gledaju sljedeći element i nikad se ne ‘sudare’. Prvo svi gledaju i+0 pa i+1, itd. Da PRAM računalo ne radi u lock-step načinu, ovaj kod nebi bio dobar.

Rješenje sa operacijom definiranom na predavanju i korištenjem scana by Bikush:

```
// pomoćna operacija koju je definirao Jakobović na satu
X(a,b)
     AKO (b==0) VRATI 0;
     INAČE VRATI (a+b);
KRAJ

// algoritam

r[] = X_SCAN(p);
max = MAX_REDUCE(r);
AKO (max >= 4)
    ispiši("DA");
INAČE
    ispiši("NE");

// primjer    p = [0 1 1 0 1 1 1 1 0]
// nakon scan r = [0 1 2 0 1 2 3 4 0]
// rješenje max_reduce(r) je najveći broj uzastopnih jedinica u p
```

Da. to je dobro.

Aj mi samo reci kak nam je sad zadatak logaritamske složenosti?

Operacija REDUCE i SCAN su logaritamske slozenosti, a ostali dijelovi koda su O(1) (paralelno blokovi).

2. (10 bodova)

Paralelni algoritam iterativno računa elemente matrice. Nova vrijednost elemenata računa se pomoću vrijednosti neposrednih elemenata gore i lijevo s tim da matrica ima spojene sve vanjske bridove (npr. vrijednost elemenata A[1,1] računa se pomoću A[N, 1] i A[1, N]). Trošak

računanja jednog elementa iznosi t_c.Izrazite izvođenje jedne iteracije na P procesora, te izoučinkovitost ako je matrica na procesore podijeljena:

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

broj poruka po procesoru: 1

trajanje izvođenja: $t_c \cdot N^2/P + (t_s + t_w \cdot N)$

izoučinkovitost: $O(P^2)$

b) po podmatricama jednake veličine:

broj poruka po procesoru: 2

trajanje izvođenja: $t_c \cdot N^2/P + 2 \cdot \left(t_s + t_w \cdot N/\sqrt{P}\right)$

izoučinkovitost: $O(P)$

3. (6 bodova)

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) 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.

```
// komunikacijska struktura hiperkocke
ZA (i = 0 DO logn - 1)
      dest_id = id XOR 2^i;
      AKO (dest_id > id)
            SEND(x, dest_id);
            RECV(drugi_podatak, dest_id);
      INAČE
            RECV(drugi_podatak, dest_id);
            SEND(x, dest_id);
      AKO (drugi_podatak < x)
            x = drugi_podatak;
// dodatan AKO uvjet je ubačen kako potencijalno ne bi došlo do
potpunog zastoja (ovisno o implementaciji)
```

Mislim da je ovo nepotrebno zakomplicirano. Nigdje se ne spominju stvari s implementacijama (ako si mislio/la na send koji čeka da se primi poruka). Tu treba obican butterfly, odnosno izbacit

linije 4,7,8 i 9

4. (4 boda)

Napišite primjer APRAM programa za tri procesora. Upotrijebite sve četiri vrste instrukcija.

Instrukcije APRAM:

- globalno čitanje - `čitaj`
- globalno pisanje - `piši`
- lokalna operacija - `operacija`
- sinkronizacija svih procesora - `sinkronizacija`

Unutar istog asinkronog odsječka samo jedan procesor može pristupiti istoj memorijskoj lokaciji.

Rješenje je slično primjeru u skripti na strani 25.

	procesor 1	**procesor 2**	**procesor 3**
odsječak 1	čitaj A čitaj B operacija piši B sinkronizacija	čitaj C operacija piši C sinkronizacija	čitaj D operacija piši D sinkronizacija
odsječak 2	 sinkronizacija	čitaj B čitaj D operacija piši D sinkronizacija	čitaj A operacija piši A sinkronizacija
odsječak 3	čitaj D operacija sinkronizacija	 sinkronizacija	čitaj B operacija sinkronizacija
	itd.	itd.	itd.

5. (4 boda)

Trajanje nekog programa dano je u dvije komponente: dio programa koji se mora izvoditi slijedno ima trajanje 100 N, dio koji se može idealno paralelizirati ima trajanje N^2 (idealna paralelizacija uz P procesora skraćuje trajanje P puta).

a) Simbolički opišite ubrzanje toga algoritma po Amdallovom zakonu ako je N = 100.

b) Izračunajte ubrzanje ako je N = 100 i P = 10.

Sažetak prijašnjih rješenja - točno rješenje (maxur):

Formula za ubrzanje(Amdalov zakon):
Ubrzanje = 1/ (Serijski_Udio + Paralelni_Udio/Broj_Procesora)

Udio znači postotak nekog posla u ukupnom poslu (vrijednost između 0 i 1). U zadatku imamo zadana vremena serijskog posla(Serijsko_Vrijeme) 100N i paralelnog posla(Paralelno_Vrijeme) N^2.
Prvo trebamo izračunati udjele:

Ukupno_Vrijeme = Serijsko_Vrijeme + Paralelno_Vrijeme
Serijski_Udio = Serijsko_Vrijeme / Ukupno_Vrijeme
Paralelni_Udio = Paralelno_Vrijeme / Ukupno_Vrijeme

Kad uvrstimo zadane vrijednosti dobijemo:

Ukupno_Vrijeme = 100N + N^2
Serijski_Udio = 100 / (100 + N)
Paralelni_Udio = N / (100 + N)

U **a)** dijelu zadatka kaže da je N=100. Kad uvrstimo gore dobijemo:

Serijski_Udio = 0.5
Paralelni_Udio = 0.5

Kad uvrstimo u Amdalov zakon uz varijablu Broj_Procesora:

Ubrzanje = 1/ (0.5 + 0.5/Broj_Procesora).

Kad se izraz sredi i stavimo P = Broj_Procesora, dobijemo:

Ubrzanje = 2P / (P+1)

U **b)** dijelu zadatka uvrstimo P =10 i dobijemo: Ubrzanje = 20/11 = 1.82.

6. (12 bodova)

- Kompozicija modula u paralelnim programima može biti **slijedna, paralelna ili zajednička**.
- MPI mehanizam modula u paralelnim programima omogućava izvedbu **slijedne i paralelne kompozicije modula**.
- MPI mehanizam dijeljenja komunikatora omogućava izvedbu **paralelne** kompozicije modula u paralelnom programu.
- Ubrzanje veće od linearnog naziva se **superlinearno**.
- Funkcija izoučinkovitosti pokazuje kako se treba mijenjati **veličina posla** uz povećanje broja procesora kako bi **učinkovitost ostala jednaka.**
- Ukoliko se ubrzanje programa mjeri u odnosu na isti program pokrenut na jednom procesoru, radi se o **relativnom** ubrzanju.
- Prilikom istodobnog čitanja iste memorijske lokacije u CRCW PRAM računalu, svaki procesor će pročitati **jednaku** vrijednost.
- U APRAM računalu, unutar istog asinkronog odsječka samo **jedan** procesor može pristupiti **istoj globalnoj** memorijskoj lokaciji.
- Izraz koji opisuje trajanje slanja jedne poruke duljine L riječi u jednostavnom modelu komunikacije je $\mathbf{T_{msg} = t_s + t_wL}$ **(t_s - postavljanje poruke, t_w - prijenos jedne riječi)**

TOČNO - NETOČNO

- Prilikom pridruživanja zadataka procesorima, zadatke koji se izvode neovisno poželjno je pridružiti istom procesoru. **NETOČNO**
- Jednom procesoru može biti dodijeljeno više MPI procesa. **TOČNO**
- Uz povećanje količine računanja i nepromijenjene ostale elemente, učinkovitost programa opada. **NETOČNO**
- Trajanje izvođenja paralelnog programa je po definiciji neovisno o promatranom procesoru. **TOČNO**
- Sitno zrnata podjela posla podrazumijeva malu količinu komunikacije u odnosu na veću količinu računanja. **NETOČNO**

7. (2 boda)

Nacrtajte podjelu sedam zadataka na četiri procesora cikličnim pridruživanjem.

Procesor	Zadaci
0	0, 4
1	1, 5
2	2, 6
3	3