Projektovanje paralelnih algoritama I

- Uvod
- Osnove dinamičke paralelizacije

Uvod

- PLATFORMA ZA DINAMIČKU PARALELIZACIJU
  - Omogućava specificiranje paralelizma u aplikaciji
  - Bez brige o komunikacionim protokolima, uravnoteženju opterećenja i drugim problemima
- Konkurentna platforma sadrži raspoređivač
  - Koji uravnotežuje opterećenje procesorskih jezgara
- Platforma podržava dve apstrakcije:
  - UGNJEŽDENI PARALELIZAM: podprogram se može pokrenuti kao paralelna aktivost (spawn)
  - PARALELNE PETLJE: iteracije se mogu izvršavati paralelno

Osnovni model dinamičke paralelizacije (1/2)

- Projektant definiše logički paralelizam u aplikaciji
  - *Niti* unutar platforme raspoređuju i uravnotežuju opterećenje same između sebe
- Ovaj model nudi sledeće 4 važne prednosti:
  - Pseudo kod sa samo tri nove ključne reči: **parallel**, **spawn** i **sync**. Njihovim brisanjem iz pseudo koda paralelnog algoritma ⇔ serijski algoritam, tzv. SERIJALIZACIJA paralelnog algoritma
  - Obezbeđuje kvantifikaciju paralelizma zasnovanu na pojmovima RAD i RASPON

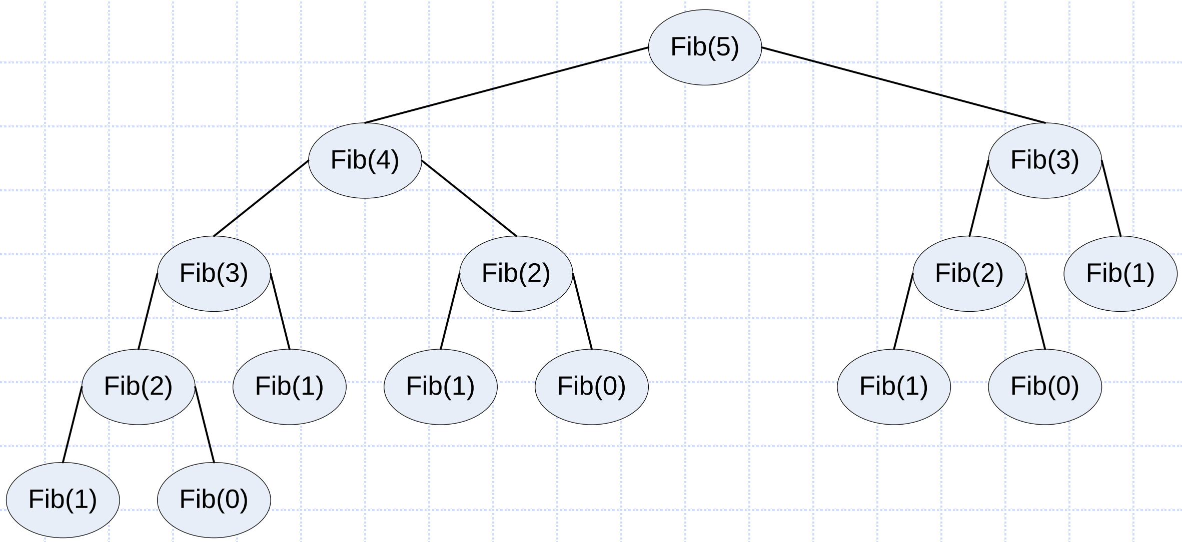
Osnovni model dinamičke paralelizacije (2/2)

- (važne prednosti modela, nastavak...):
  - Mnogi paralelni algoritmi proističu iz prakse podeli-i-zavladaj. Oni podležu analizi pomoću rešavanja rekurencija (kao i serijski algoritmi tog tipa)
  - Moderne konkurentne platforme podržavaju neku od varijanti dinamičke paralelizacije. Npr. Cilk, Cilk++, OpenMP, TPL (Task Parallel Library) i TBB (Threading Building Blocks)

Osnovi dinamičke paralelizacije

- Primer rekurz. računanja Fibonačijevih brojeva:
  - $F_0 = 0$; $F_1 = 1$; $F_i = F_{i-1} + F_{i-2}$ za $i \geq 2$
  - Serijski algoritam i stablo rekurzivnih instanci Fib(*n*):

```
Fib(n)
1.if n ≤ 1
2.       return n
3.else x = Fib(n - 1)
4.       y = Fib(n - 2)
5.       return x + y
```



- Nedostatak ovog algoritma: obavlja dosta istog posla više puta
- Npr. Fib(5) poziva Fib(4) i Fib(3), Fib(4) takođe poziva Fib(3)

Analiza serijskog algoritma

- Rekurentna jednačina za proceduru Fib(n):

  $T(n) = T(n-1) + T(n-2) + \Theta(1)$

  - Rešenje: $T(n) = \Theta(F_n) = \Theta(\phi^n)$, gde je $\phi = (1+5^{1/2})/2$ tzv. zlatni odnos

- Provera metodom zamene:
  - Induktivna hipoteza: $T(n) \leq aF_n - b$, a i b konstante
  - Zamenom dobijamo:

$$
\begin{aligned}
T(n) &\leq (aF_{n-1} - b) + (aF_{n-2} - b) + \Theta(1) \\
&= a(F_{n-1} + F_{n-2}) - 2b + \Theta(1) \\
&= aF_n - b - (b - \Theta(1)) \\
&\leq aF_n - b
\end{aligned}
$$

Paralelni algoritam za računanje F_n

- Pseudo kod se proširuje dodavanjem ključnih reči **spawn** i **sync**, rezultat je P-Fib(*n*)
  - Uklanjanjem **spawn** i **sync** iz P-Fib(*n*) dobija se SERIJALIZACIJA – serijski kod koji rešava isti problem

```
P-Fib(n)
1.if n ≤ 1
2.      return n
3.else x = spawn P-Fib(n - 1)
4.      y = P-Fib(n - 2)
5.      sync
6.      return x + y
```

- Ugnježđeni paralelizam se dešava uvek kada ključna reč **spawn** prethodi pozivu procedure, kao npr. u liniji 3
- **spawn** izražava logički paralelizam u računanju: ukazuje na delove koji mogu ići u paralleli, simbol: ||
- Prilikom izvršenja, raspoređivač određuje koja podračunanja se zaista izvršavaju paralelno

Ugnježdeni paralelizam

```
P-Fib(n)
1.if n ≤ 1
2.      return n
3.else x = spawn P-Fib(n - 1)
4.      y = P-Fib(n - 2)
5.      sync
6.      return x + y
```

- Semantika ključne reči **spawn** se razlikuje od običnog poziva procedure
- Instanca procedure koja izvrši **spawn**, tzv. predak, može da se izvršava u paraleli sa novom instancom, tzv. potomak
- Npr. dok potomak računa P-Fib(n - 1), predak može paralelno računati P-Fib(n - 2) u liniji 4
- P-Fib ne može koristiti vrednost x koju vraća izmrešćeni potomak sve dok ne izvrši **sync** u liniji 5

Model paralelnog izvršavanja (1/3)

- Graf računanja $G = (V, E)$
  - Čvorovi u V su instrukcije
  - Grane u E pretstavljaju zavisnosti između instrukcija
  - $(u, v) \in E$ znači da se u mora izvršiti pre v
- Serijski iskazi se mogu grupisati u jednu liniju (eng. strand) izvršenja
- Instr. za kontrolu || izvršenja se ne uključuju u linije već se pretstavljaju u strukturi grafa
  - Dakle, skup V formira skup linija, dok skup usmerenih grana iz E pretstavlja zavisnosti između njih

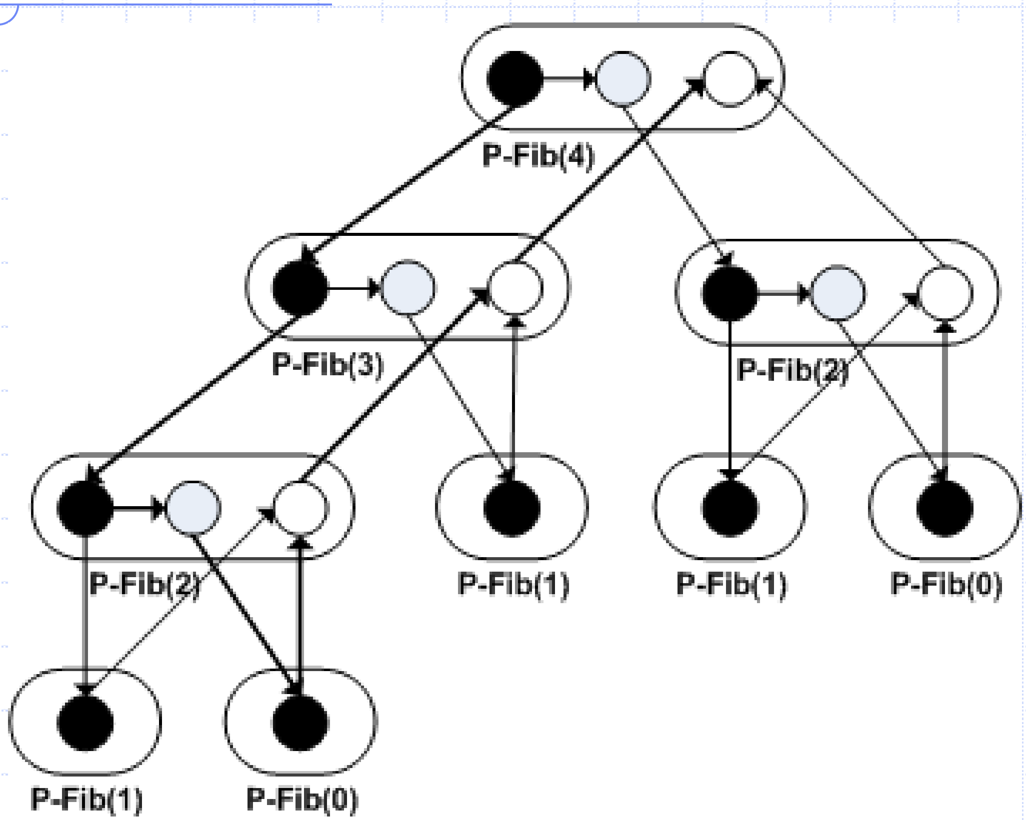
Model paralelnog izvršavanja (2/3)

- Ako u *G* postoji usmerena putanja od linije *u* do linije *v*, dve linije su logički u rednoj vezi
  - U suprotnom, linije *u* i *v* su logički u paralelnoj vezi
- Graf linija izvršenja je ugrađen u stablo instanci procedure
  - On je uvećan detalj tog stabla (zoom-in)
  - Npr. u prethodnom stablu instanci procedure za poziv P-Fib(5), možemo uvećati deo za računanje P-Fib(4)

Model paralelnog izvršavanja (3/3)

- 4 vrste grana grafa računanja:
  - GRANA NASTAVKA (*u*, *u′*) se crta udesno, povezuje liniju *u* sa njenim sledbenikom *u′* unutar iste instance procedure
  - GRANA MREŠĆENJA (*u*, *v*) se crta nadole, i pokazuje da je linija izvršenja *u* izmrestila liniju *v*
  - GRANA POZIVA se crta takođe nadole, i pretstavlja običan poziv procedure (ali ne indukuje nastavak)
  - GRANA POVRATKA (*u*, *x*) se crta na gore, i pokazuje da se linija izvršenja *u* vraća njenoj pozivajućoj proceduri *x*

Graf računanja za poziv Fib(4)



- Ovalni simbol: jedna linija
- Crni čvor: osnovni slučaj ili deo instance procedure do tačke mreščenja
- Osenčeni čvor: deo procedure koji poziva P-Fib(*n*-2), od linije 4 do iskaza **sync**
- Beli čvor: deo procedure iza iskaza **sync**
- Kritična putanja u grafu je označena punijim strelicama

Idealan paralelni računar

- Skup procesora i sekvencijalno konzistentne (SK) deljene memorije
  - SK: više istovremenih operacija R i W, kao kada bi se obavljala po jedan R/W od strane jednog procesora
  - Linearan redosled, koji očuvava redosled grafa

- Pretpostavlja se da su svi procesori iste snage i zanemaruje se cena raspoređivanja

Mere performanse

- Dve mere efikasnost paralelnog algoritma:
  - RAD: ukupno vreme računanje na jednom procesoru. Ako je vreme za svaki čvor 1, rad je = br. čvorova
  - RASPON: najveće vreme potrebno da se izvrše linije duž bilo kog puta. Ako je vreme za svaki čvor 1, raspon je = br. čvorova duž najduže putanje u grafu
- Def: KRITIČNA PUTANJA grafa je najduža putanja u grafu
- Primer: graf Fib(4) ima 17 čvorova i 8 je na kritičnoj putanji, ako je vreme za svaki čvor 1, rad=17, raspon=8

Vremena T_P, T_1 i T_∞

- Stvarno vreme izvršenja paralelnog algoritma:
  - Koliko je procesora raspoloživo?
  - Kako raspoređivač dodeljuje procesore linijama?
- Oznake
  - T_P vreme na P procesora
  - T_1 vreme na jednom procesoru (= rad)
  - T_∞ vreme za neograničen broj procesora (= raspon, jer bi se svaka linija izvršavala na svom procesoru)
- Rad i raspon obezbeđuju donje granice za vreme izvršenja T_P – zakon rada i zakon raspona

Zakoni rada i raspona, ubrzanje

- ZAKON RADA: $T_P \geq T_1 / P$
- ZAKON RASPONA: $T_P \geq T_\infty$
- Def: UBRZANJE na P procesora je odnos T_1 / T_P
  - Govori koliko puta je brže izvršenje na P procesora
  - Ako je $T_1 / T_P = \Theta(P)$, to je LINEARNO UBRZANJE
  - Ako je $T_1 / T_P = P$, to je SAVRŠENO LINEARNO UBRZANJE

Paralelizam

- Def: PARALELIZAM paralelnog algoritma je odnos T_1 / T_∞. Tri perspektive:
  - Kao odnos: srednja količina rada, koja se može obaviti u paraleli
  - Kao gornja granica, najveće moguće ubrzanje
  - Ograničenje savršenog linearnog ubrzanja (max br procesora za savršeno linearno ubrzanje)
- Npr, rad T_1=17, raspon T_∞=8 $\Rightarrow$ paralelizam je $T_1 / T_\infty = 17/8 = 2.125$
  - Posledica: nemoguće ostvariti ubrzanje mnogo veće od duplog (označava se kao 2x)

Labavost paralelizma

- Def: To je odnos $(T_1 / T_\infty)/P = T_1 / (P\, T_\infty)$
  - Smisao: faktor sa kojim paralelizam algoritma prevazilazi broj procesora
  - Ako je labavost manja od 1, ne može se ostvari savršeno linearno ubrzanje
    - Iz $T_1 / (P\, T_\infty) < 1 \Rightarrow T_1 / T_P \leq T_1 / T_\infty < P$
    - Kako se labavost smanjuje od 1 ka 0, ubrzanje divergira sve dalje od savršenog
  - Ako pak je labavost veća od 1, ograničavajući faktor je rad po procesoru
  - Kako se labavost povećava od 1, ubrzanje sve bliže savršenom

Raspoređivanje

- Apstrakcija: raspoređivač konkurentne platforme direktno preslikava linije na procesore
  - U stvarnosti: Raspoređivač preslikava linije na statičke niti, a OS raspoređuje niti na procesore
- POHLEPNI RASPOREĐIVAČI dodeljuju što je moguće više linija u svakom koraku. Korak:
  - POTPUN: barem P linija $\Rightarrow$ P linija na izvršenje
  - NEPOTPUN: manje od P linija $\Rightarrow$ sve na izvršenje
- Iz zakona rada min $T_P = T_1 / P$, iz zakona raspona min $T_P = T_\infty$. Koliko je max T_P?

Teorema o gornjoj granici T_P

- Na P procesora, pohlepni raspoređivač izvršava paralelni algoritam sa radom T_1 i rasponom T_∞ u vremenu :

$$T_P \leq T_1/P + T_\infty$$

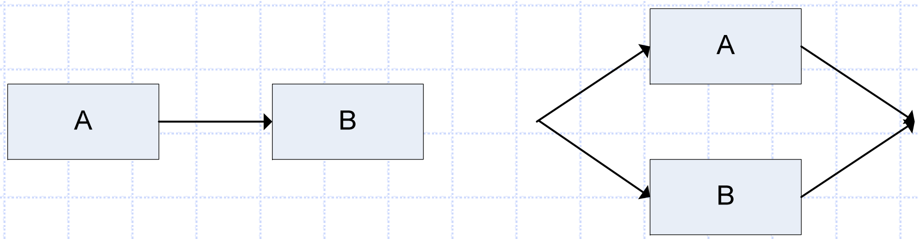
  - I posledica: T_P nikada nije više od 2 puta veće od optimalnog vremena T_P*: $T_P \leq 2\ T_P$*
  - II posledica: ako je $P << T_1 / T_\infty \Rightarrow$ vreme $T_P \approx T_1/P$, ili ekvivalentno, da je ubrzanje približno jednako sa P

Analiza paralelnih algoritama

- Cilj analize: odrediti granice T_P za različite br. P
  - Analiza rada $\Leftrightarrow$ analiza SERIJALIZACIJE
  - Analiza raspona $\Leftrightarrow$ analiza KRITIČNOG PUTA
- Analiza raspona razlikuje 2 slučaja:
  - Ako su dva podgrafa A i B povezana redno:

    $T_\infty(A \cup B) = T_\infty(A) + T_\infty(B)$
  - Ako su dva podgrafa A i B povezana paralelno:

    $T_\infty(A \cup B) = \max(T_\infty(A), T_\infty(B))$



(a) (b)

Primer: Analiza P-Fib(n)

- Rad: pošto je Fib serijalizacija od P-Fib:

  $T_1(n) = T(n) = \Theta(\phi^n)$

- Raspon: pošto unutar P-Fib(n) postoji paralelna veza P-Fib($n-1$) i P-Fib($n-2$):

  $$T_\infty(n) = \max(T_\infty(n-1), T_\infty(n-2)) + \Theta(1)$$
  $$= T_\infty(n-1) + \Theta(1)$$

  - Rešenje ove jednačine je $T_\infty(n) = \Theta(n)$

- Paralelizam P-Fib(n) je:

  $T_1(n)/T_\infty(n) = \Theta(\phi^n/n)$

  - Već za malo n, savršeno ubrzanje, čak i na najvećim paralelnim računarima (zbog velike labavosti)

Paralelne petlje

- Petlje čije sve iteracije mogu da se izvršavaju ||
  - Obično se paralelizuju pomoću **parallel for**
  - Primer: moženje matrice *A* sa vektorom *x* :

$$y_i = \Sigma_{j=1..n} a_{ij} x_j , \quad i = 1, 2, ..., n$$

```
Mat-Vec(A, x)
1.n = A.rows
2.neka je y novi vektor dužine n
3.parallel for i = 1 to n
4.    y_i = 0
5.parallel for i = 1 to n
6.    for j = 1 to n
7.        y_i = y_i + a_ij x_j
8.return y
```

Realizacija paralelne petlje

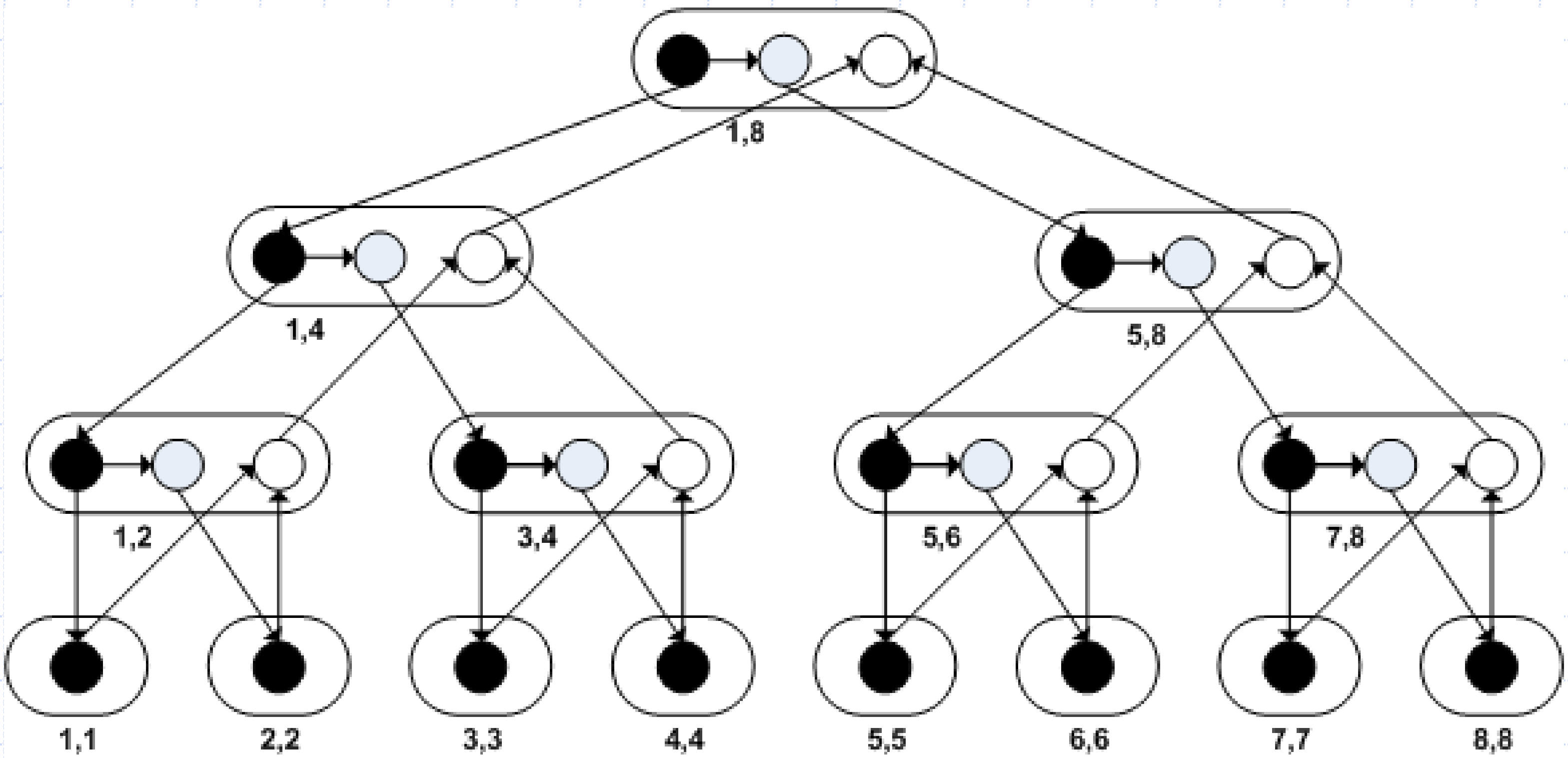
Mat-Vec(A, x)
1.n = A.rows
2.neka je y novi vector dužine n
3.parallel for i = 1 **to** n
4. $y_i = 0$
5.parallel for i = 1 **to** n
6.   **for** j = 1 **to** n
7. $y_i = y_i + a_{ij} x_j$
8.return y

- Kompajler realizuje svaku **parallel for** petlju kao proceduru zasnovanu na pristupu podeli-i-zavladaj, koja koristi ugnježdeni paralelizam

Mat-Vec-Main-Loop(A, x, y, n, i, i')
1.if $i == i'$
2.   **for** j = 1 to n
3. $y_i = y_i + a_{ij} x_j$
4.else $mid = \lfloor (i + i')/2 \rfloor$
5.   **spawn** Mat-Vec-Main-Loop(A, x, y, n, i, mid)
6.   Mat-Vec-Main-Loop(A, x, y, n, mid+1, i')
7.   **sync**

Binarno stablo računanja za proceduru Mat-Vec-Main-Loop

- Graf za poziv Mat-Vec-Main-Loop(A,x,y,8,1,8).
  - dva broja ispod linija su vrednosti indeksa: i , i'



Analiza procedure Mat-Vec (1/2)

- Rad $T_1(n)$:
  - Serijalizacija: **parallel for** petlje se zamene običnim **for** petljama
  - $T_1(n) = \Theta(n^2)$, pošto je kvadratno vremene izvršenja dvostruke ugnježdene petlje dominantno
- Koliki je gubitak vremena zbog mrešćenja?
  - Br. unutrašnjih čvorova je za 1 manji od broja listova
  - Svaki čvor uzima $\Theta(1)$
  - Svaki list odgovara jednoj iteraciji, koja uzima $\Theta(1)$
  - Sve zajedno to je $\Theta(n)$, što $\Theta(n^2)$ absorbuje

Analiza procedure Mat-Vec (2/2)

- Raspon $T_\infty(n)$:
  - Pošto je dubina rekurzivnog pozivanja logaritamska funkcija broja iteracija, za paralelnu petlju sa n iteracija, u kojoj i-ta iteracija ima raspon $iter_\infty(i)$:

    $$T_\infty(n) = \Theta(\lg n) + \max_{1 \le i \le n} iter_\infty(i)$$

    - Paralelna petlja u linijama 3-4 ima raspon $\Theta(\lg n)$
    - Raspon dve ugnježdene petlje u linijama 5-7 je $\Theta(n)$
    - Raspon preostalog koda u toj proceduri je konstantan
    - Raspon cele procedure je $\Theta(n)$

  - Kako je rad $\Theta(n^2)$, paralelizam je $\Theta(n^2)/\Theta(n) = \Theta(n)$

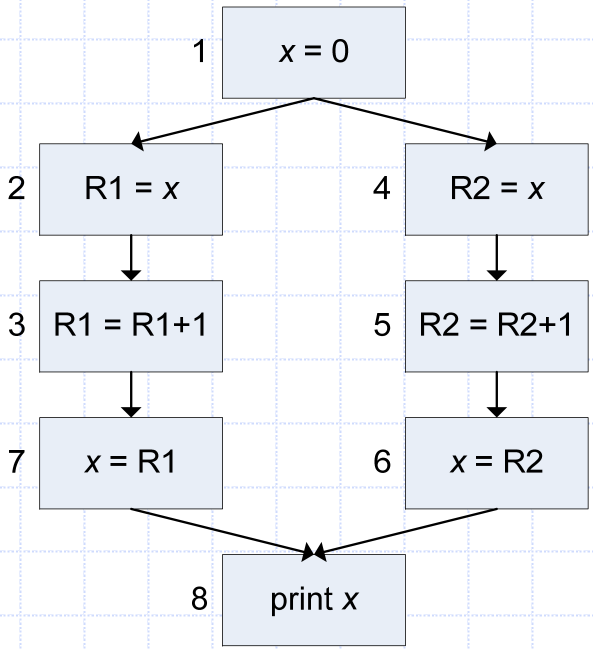
Trka do podataka

- Paralelni algoritam je DETERMINISTIČAN, ako je njegovo ponašanje uvek isto za isti ulaz
- TRKA DO PODATAKA se dešava između dve logički paralelne instrukcije koje pristupaju istoj memorijskoj lokaciji i bar jedna od tih instrukcija upisuje u tu lokaciju. Primer:

```
Race-Example()
1.x = 0
2.parallel for i = 1 to 2
3.    x = x + 1
4.print x
```

- Anomalija se dešava zato što se sadržaj *x* ne uvećava za 1 odjednom, već nizom instrukcija:
  1. Učitaj *x* iz memorije u registar
  2. Uvećaj sadržaj tog registra za 1
  3. Upiši registar u *x* u memoriji

Ilustracija trke do podataka



(a)

Korak	x	R1	R2
1	0	-	-
2	0	0	-
3	0	1	-
4	0	1	0
5	0	1	1
6	1	1	1
7	1	1	1

(b)

- U prikazanom redosledu instrukcija procedura pogrešno štampa 1.
- Problem: Mnoga izvršenja ne otkrivaju grešku!
- Npr. redosled izvršenja <1,2,3,7,4,5,6,8> ili <1,4,5,6,2,3,7,8> proizvodi ispravan rezultat 2.

Rešenje trke do podataka

- Postoje razni načini
  - Npr. brave za međusobno isključivanje, itd.
- Najbolje rešenje:
  - U konstrukciji sa **parallel for** sve iteracije treba da budu nezavisne
  - Između **spawn i sync**, programski kod potomka treba da bude nezavistan od koda njegovog pretka
    - Uključujući kod dodatno izmrešćenih ili pozvanih potomaka!

Oprezno! Jako je lako pogrešiti!

- Neispravna realizacija paralelnog množenja matrice vektorom, sa rasponom $\Theta(\lg n)$, tako što paralelizuje unutrašnju **for** petlju:

Mat-Vec-Wrong(A, x)
1. $n = A.\text{rows}$
2. neka je y novi vector dužine n
3. **parallel for** $i = 1$ **to** n
4. $\quad y_i = 0$
5. **parallel for** $i = 1$ **to** n
6. $\quad$ **parallel for** $j = 1$ **to** n
7. $\qquad y_i = y_i + a_{ij} x_j$
8. **return** y

- Neispravna zbog trke prilikom ažuriranja y_i u liniji 7, koja se izvršavaju konkurentno za svih n vrednosti indeksa j
- Domaći: napraviti ispravnu realizaciju koja ostvaruje raspon $\Theta(\lg n)$

Lekcija iz paralelizma (1/2)

- Istinita priča
  - Desila se prilikom razvoja programa za igranje šaha, pod nazivom Sokrates
  - Razvoj na 32 procesora, ciljni računar 512 procesora
  - Jedna optimizacija je smanjila vreme izvršenja važnog referentnog testa sa T_{32} = 65s na T_{32}' = 40s
  - Na osnovu rada i raspona zaključeno da će ta verzija, koja je bila brža na 32 procesora, ustvari biti sporija na ciljnom računaru sa 512 procesora
  - Epilog: napustili su tu optimizaciju

Lekcija iz paralelizma (2/2)

- Proračun:
  - Originalna verzija T_1=2048s i raspon T_∞=1s
  - Ako se uzme $T_P = T_1/P + T_\infty$
  - $T_{32} = 2048/32 + 1 = 65$
  - $T_{32}' = 1024/32 + 8 = 40$
  - $T_{512} = 2048/512 + 1 = 5$s
  - $T_{512}' = 1024/512 + 8 = 10$s
  - Dva puta sporije na 512 procesora!
- Pouka: Bolje računati rad i raspon nego osloniti se samo na merenja