

OpenMP

Šta je OpenMP

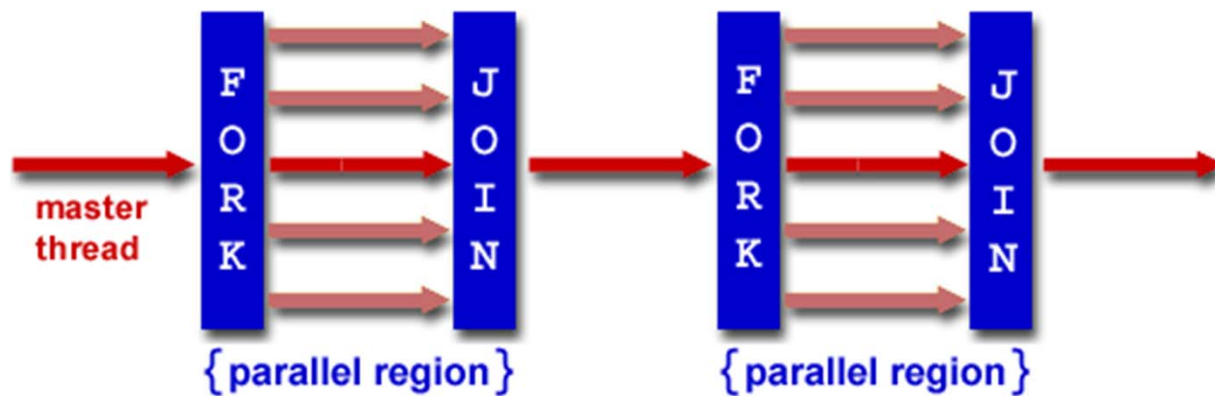
- * OpenMP predstavlja skup kompajlerskih direktiva (u C-u se one zovu pragma) koje omogućavaju paralelizaciju programa za sisteme sa deljivom memorijom (shared memory), kao što su npr. multicore procesori
- * OpenMP se sastoji od
 - Skupa direktiva
 - Skupa (run-time) bibliotečkih funkcija
 - Skupa promenljivih okruženja (environment variables)
- * Ove tri stvari čine API za paralelno programiranje na sistemima sa deljivom memorijom.
 - OpenMP je nezavistan od hardvera i OS.
 - Postoji za sve glavne verzije UNIX i Windows OS

OpenMP - programski model

* OpenMP koristi fork-join model paralelnog izvršenja.

- Svaki OpenMP program počinje sa jednom niti izvršenja (master thread - glavna nit).
- U fork-join modelu master nit se izvršava sekvencijalno dok ne naiđe na direktivu kojom se definiše paralelni region, kada se kreira tim paralelnih niti (FORK - viljuška) koje dalje nastavljaju paralelno da se izvršavaju.
- Kada tim niti okonča izvršenje paralelnog regiona, sinhronizuju se i okončavaju sa izvršenjem, nakon čega ostaje samo master thread koji se izvršava sekvencijalno (JOIN).
- Na kraju svakog paralelnog regiona se nalazi implicitna tačka sinhronizacije, tzv. barijera.
 - Kada se koristi ova vrsta sinhronizacije ni jedna nit ne može da nasatvi sa izvršenjem sve dok sve niti ne stignu do barijere.
- Niti u OpenMP komuniciraju preko deljivih promenljivih

Programski model: Fork-join model izvršenja



OpenMP - osobine

* OpenMP direktive su specijalno formatirani komentari ili pragme koje se primenjuju na niz naredbi koji sledi iza njih.

- Najveći broj direktiva se primenjuje na strukturni blok naredbi.

- Strukturni blok predstavlja skup izvršnih naredbi sa jednim ulazom na vrhu i jednim izlazom na dnu (kraju) bloka.

* OpenMP omogućava programeru da

- kreira tim niti za paralelno izvršenje

- specificira kako se vrši podela posla između niti u timu

- deklariše deljive i lokalne promenljive za niti

- sinhronizuje niti i omogućí da se neke operacije obavljaju ekskluzivno, tj. uzajamno isključivo

Kreiranje tima niti

- * Tim niti se kreira da bi se izvršile naredbe koje se nalaze u paralelnom regionu.
- * Da bi se kreirao tim niti programer jednostavno specificira paralelni region koristeći direktivu **parallel**
 - Uz ovu direktivu moguće je navesti dodatne informacije o kojima će biti reči kasnije.
- * Na kraju svakog paralelnog regiona se nalazi implicitna tačka sinhronizacije, tzv. barijera.
 - Kada se koristi ova vrsta sinhronizacije ni jedna nit ne može da nastavi sa izvršenjem sve dok sve niti ne stignu do barijere

Podela posla između niti

- * Ako programer ne navede kako će se izvršiti podela posla između niti u paralelnom regionu, svaka nit će izvršiti sve naredbe koje se nalaze u paralelnom regionu.
 - Ovakav prilaz očigledno ne bi doveo do ubrzanja izvršenja programa.
- * OpenMP direktive za podelu posla omogućavaju programeru da odredi kako će se izračunavanje u strukturnom bloku podeliti između niti u timu.
 - I kod ovih direktiva se implicitno podrazumeva barijerna sinhronizacija na kraju direktive.
 - Programer može eksplicitno ukinuti barijeru na kraju konstrukcija za podelu posla.
- * To su direktive
 - for, sections, single i task.

Memorijski model OpenMP

- * OpenMP je baziran na deljivoj memoriji i po definiciji (by default) podaci su deljivi između niti i vidljivi svim nitima.
- * Ponekad postoji potreba da izvesni podaci budu privatni za niti.
 - Npr. kada tim niti izvršava paralelnu for petlju neophodno je da svaka nit ima svoj privatni indeks petlje (iterativnu promenljivu petlje).
 - Zbog toga je indeksna promenljiva paralelne for petlje privatna za svaku nit u timu.
- * OpenMP pruža mogućnost programeru i da eksplicitno definiše koje su promenljive deljive a koje privatne korišćenjem posebnih odredbi ili klauzula (clauses).
 - Niti u OpenMP komuniciraju preko deljivih promenljivih.
- * Potrebno je obezbediti da niti zapamte privatne podatke u toku izvršenja.
 - Za ovo postoji posebna oblast u memoriji poznata kao thread stack (magacin niti).

Sinhronizacija niti

- * Sinhronizacija ili koordinacija akcija niti je nekad neophodna da bi se obezbedio korektan pristup deljivim (zajedničkim) podacima i da bi se sprečilo narušavanje podataka.
- * OpenMP nudi relativno mali skup jednostavnih sinhronizacionih mehanizama.
- * Već smo napomenuli da OpenMP usvaja implicitno postojanje barijere na kraju svake paralelne konstrukcije za podelu posla.
 - Kod ovog vida sinhronizacije zahteva se da sve niti stignu do barijere pre nego što nastave dalje sa izvršenjem.
 - Sinhronizaciju samo određenog podskupa niti je teže postići i zahteva veći napor jer OpenMP ne sadrži eksplicitnu podršku za to.
- * Ponekad je neophodno obezbediti da samo jedna nit u jednom trenutku izvršava deo programskog koda.
 - OpenMP ima nekoliko mehanizama koji omogućavaju ovakav vid sinhronizacije.
- * Za sinhronizaciju niti, pored barrier, OpenMP nudi i direktive:
 - ordered, critical, atomic, lock, master.

Koliko niti je u timu?

- * Za neke aplikacije je važno da može da se upravlja brojem niti koje izvršavaju paralelni region.
- * OpenMP omogućava programeru da specificira ovaj broj
 - pre izvršenja programa korišćenjem odgovarajuće promenljive okruženja (environment variable),
 - nakon početka izvršenja korišćenjem izvršne bibliotečke funkcije, ili
 - na početku paralelnog regiona korišćenjem odgovarajuće odredbe (klauzule).
- * Ako se ovo ne uradi, onda broj niti zavisi od implementacije.
- * OpenMP svakoj niti u timu dodeljuje jedinstveni identifikator koji se kreće u garnicama od 0 do broj niti-1.

OpenMP - detalji

* Sintaksa OpenMP direktiva je sledeća

#pragma omp ime_direktive [*odredba*[[*,*] *odredba*]....]

- *omp* ključna reč govori da je reč o OpenMP pragmi koje kompajlira samo OpenMP kompajler, a ignorišu ne OpenMP kompajleri
- *ime_direktive* ime direktive (npr. parallel, for, section,)
- *odredbe* (klauzule, eng. clauses) se koriste da daju dodatne informacije direktivi
 - (npr **#pragma omp for** private (a).
 - Ovde je private odredba koja kaže da je promenljiva a privatna za svaku nit u timu.)

OpenMP direktive

* U OpenMP su na raspolaganju sledeće direktive

- Parallel direktiva
- Direktive za podelu posla
 - for
 - section
 - single
 - task
- Direktive za sinhronizaciju niti
 - Barrier
 - Ordered
 - Critical
 - Atomic
 - Locks
 - Master

* Svaka direktiva opciono može sadržati i veći broj odrebi (klauzula)

Direktiva parallel

- * Osnovna direktiva u OpenMP je parallel direktiva
- * Sitaksa
 - `#pragma omp parallel [odredba[,] odredba]...`
 - strukturni blok
- * Ova direktiva se koristi da definiše izračunavanje koje treba da se obavi paralelno
- * Ovom direktivom se kreira tim niti i definiše paralelni region u kome će taj tim raditi.
 - Paralelni region se završava na kraju strukturnog bloka.
 - U najvećem broju slučajeva strukturni blok je ograničen parom zagrada { }.
 - Kada nit programa dođe do ove direktive kreira se tim niti da izvrši paralelni region.
 - Ova direktiva obezbeđuje da se izračunavanja obave paralelno ali ne obavlja raspodelu (distribuciju) posla unutar regiona između tima niti.
 - Ako programer ne upotrebi odgovarajuću direktivu za podelu posla, ceo posao će biti repliciran.
 - Na kraju paralelnog regiona je implicitna barrier direktiva koja prisiljava niti da čekaju dok sve niti iz tima ne obave posao unutar paralelnog regiona.
 - Nit koja naiđe na parallel direktivu postaje master nit u timu niti.
 - Svakoj niti u timu se dodeljuje identifikator niti (thread id).
 - Master nit ima id=0. Identifikatori se kreću od 0 do broja niti-1
 - Bibliotečka funkcija `omp_get_thread_num()` vraća identifikator svake niti u timu.

Primer

```
#include <omp.h>
#include <stdio.h>
void main()
{
    #pragma omp parallel
    {
        int ID = omp_get_thread_num();
        printf(" hello(%d) ", ID);
        printf(" world(%d) \n", ID);
    }
}
```

* Mada se paralelni region izvršava od strane svih niti u timu, svaka nit može imati svoj put izvršenja

* Jedan od mogućih izlaza za 4 niti:

```
hello(1) hello(0) world(1)
world(0)
hello (3) hello(2) world(3)
world(2)
```

Primer2

```
#pragma omp parallel
{
    printf("The parallel region is executed by thread %d\n",
        omp_get_thread_num());

    if ( omp_get_thread_num() == 2 ) {
        printf("  Thread %d does things differently\n",
            omp_get_thread_num());
    }
} /*-- End of parallel region --*/
```

U ovom primeru će sve niti izvršiti prvu printf, ali će drugu printf izvršiti samo nit čiji je id=2.

U slučaju da se tim sastoji od 4 niti, jedan od mogućih izlaza bi bio

```
The parallel region is executed by thread 0
The parallel region is executed by thread 3
The parallel region is executed by thread 2
  Thread 2 does things differently
The parallel region is executed by thread 1
```

Moguće odredbe (klauzule) uz parallel direktivu

* Sa parallel direktivom mogu se koristiti sledeće odredbe

<code>if(<i>scalar-expression</i>)</code>	(C/C++)
<code>if(<i>scalar-logical-expression</i>)</code>	(Fortran)
<code>num_threads(<i>integer-expression</i>)</code>	(C/C++)
<code>num_threads(<i>scalar-integer-expression</i>)</code>	(Fortran)
<code>private(<i>list</i>)</code>	
<code>firstprivate(<i>list</i>)</code>	
<code>shared(<i>list</i>)</code>	
<code>default(none shared)</code>	(C/C++)
<code>default(none shared private)</code>	(Fortran)
<code>copyin(<i>list</i>)</code>	
<code>reduction(<i>operator:list</i>)</code>	(C/C++)
<code>reduction({<i>operator</i> <i>intrinsic_procedure_name</i>}:<i>list</i>)</code>	(Fortran)

- samo jedna if odredba se može pojaviti u parallel direktivi
- samo jedna num_threads odredba se može pojaviti u direktivi. Izraz mora vratiti pozitivan ceo broj

Direktive za podelu posla između niti

* definišu kako će posao u paralelnom regionu biti podeljen između niti.

- Ove direktive se moraju naći u okviru regiona koji je definisan `parallel` direktivom da bi imale efekta
- Ako se ovakvae direktive pojave u okviru sekvencijalnog regiona, jednostavno se ignorišu

* OpenMP. C/C++ ima 4 takve direktive

Functionality	Syntax in C/C++
Distribute iterations over the threads	<code>#pragma omp for</code>
Distribute independent work units	<code>#pragma omp sections</code>
Only one thread executes the code block	<code>#pragma omp single</code>

`#pragma omp task`

Bernštajnovi uslovi paralelizma

* Neka su P1 i P2 dva procesa

- Sa I_i označimo ulazne promenljive procesa P_i
- Sa O_i označimo izlazne promenljive procesa P_i

* Procesi P1 i P2 se mogu izvršavati paralelno ako

* $I_1 \cap O_2 = \emptyset$

* $I_2 \cap O_1 = \emptyset$

* $O_1 \cap O_2 = \emptyset$

* $I_1 \cap I_2$ ne mora biti parazan skup

Primer

* $I_1 : x = (a + b) / (a * b)$

* $I_2 : y = (b + c) * d$

* $I_3 : z = x^2 + (a * e)$

* read set i write set instrukcija I_1 , I_2 i I_3 dati su sa:

* $R_1 = \{a, b\}$ $W_1 = \{x\}$

* $R_2 = \{b, c, d\}$ $W_2 = \{y\}$

* $R_3 = \{x, a, e\}$ $W_3 = \{z\}$

Direktive za podelu posla (nast.)

- * Ove direktive ne kreiraju nove niti i nemaju barijeru na ulasku.
- * Po definiciji, niti čekaju na barijeri na kraju regiona kojim se definiše podela posla dok i poslednja nit ne okonča sa izvršenjem svog dela posla.
 - Programer može ovo poništiti korišćenjem *nowait* klauzule

for direktiva

- Ova direktiva uzrokuje da iteracije petlje budu izvršene paralelno.
- U C/C++ programima korišćenje ove direktive je ograničeno na brojačke petlje
- U fazi izvršenja (run-time) iteracije petlje se distribuiraju nitima.
- Sintaksa

```
#pragma omp for [clause[[,] clause]...]  
for-loop
```

- petlja koja se paralelizuje ne sme imati zavisnosti između različitih iteracija (ni loop carry ni WAR)!

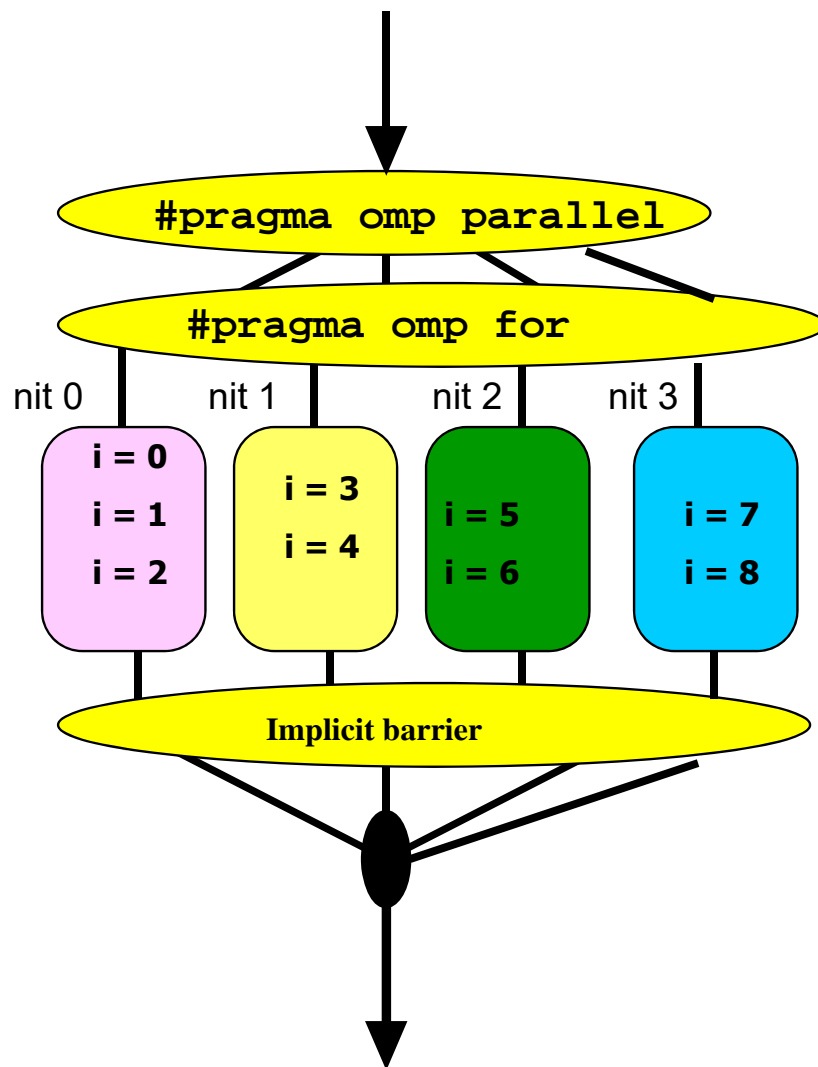
Primer

```
#pragma omp parallel shared(n) private(i)
{
    #pragma omp for
    for (i=0; i<n; i++)
        printf("Thread %d executes loop iteration %d\n",
               omp_get_thread_num(), i);
} /*-- End of parallel region --*/
```

Jedan od mogućih izlaza za n=9 i 4 niti je

```
Thread 0 executes loop iteration 0
Thread 0 executes loop iteration 1
Thread 0 executes loop iteration 2
Thread 3 executes loop iteration 7
Thread 3 executes loop iteration 8
Thread 2 executes loop iteration 5
Thread 2 executes loop iteration 6
Thread 1 executes loop iteration 3
Thread 1 executes loop iteration 4
```

Primer: podela posla između niti u for petlji



- Niti sa identifikatorima 1,2 i 3 izvršavaju po 2 iteracije petlje a nit 0 tri.
- Ako programer eksplicitno ne navede kako se vrši distribucija posla po nitima, onda će to uraditi kompajler.
- Distribucija iteracija po nitima zavisi od implementacije kompajlera.
- Moguće je da u istoj aplikaciji u različitim petljama bude različita distribucija posla po nitima.
- Pomoću *schedule* odredbe (klauzule) programer može da definiše kako se vrši distribucija posla.

Tipovi odredbi za for direktivu

```
private(list)  
firstprivate(list)  
lastprivate(list)  
reduction(operator:list) (C/C++)  
reduction({ operator | intrinsic_procedure_name } :list) (Fortran)  
ordered  
schedule (kind[, chunk_size])  
nowait
```


Section direktiva

* Sastoji se od dve direktive

- **# pragma omp sections** kojom se ukazuje na početak konstrukcije, i nekoliko
- **#pragma omp section** direktiva kojima se označva svaka pojedina sekcija koja će se izvršavati paralelno sa ostalim.
 - Svaka sekcija mora biti strukturni blok koji ne zavisi od drugih sekcija.
 - Ako ima više sekcija nego niti, onda će neke (li sve) niti izvršavati više sekcija, ali će svaka sekcija biti izvršena tačno jednom.
 - Ako ima više niti nego sekcija, višak niti biće neupošljen.
 - Dodela posla nitima je zavisna od implementacije

sections/section direktiva sintaksa

```
#pragma omp sections [clause[ [, ] clause ] ... ]  
{  
    [#pragma omp section ]  
        structured block  
    [#pragma omp section  
        structured block ]  
    ...  
}
```

* najčešće se koristi za paralelno izvršenje funkcija ili potprograma

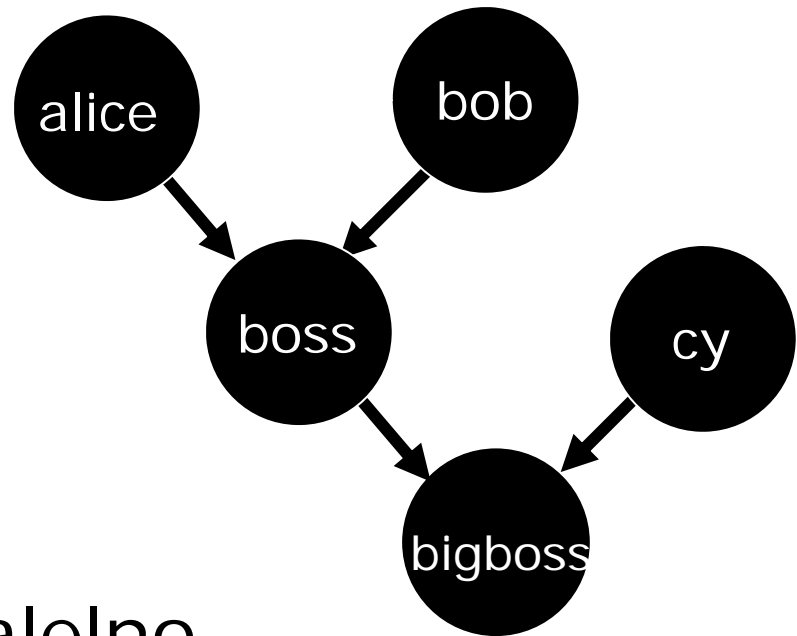
- prva section direktiva može biti izostavljena

paralelizam na nivou funkcija

```
a = alice();  
b = bob();  
s = boss(a, b);  
c = cy();  
printf ("%6.2f\n",  
        bigboss(s,c));
```

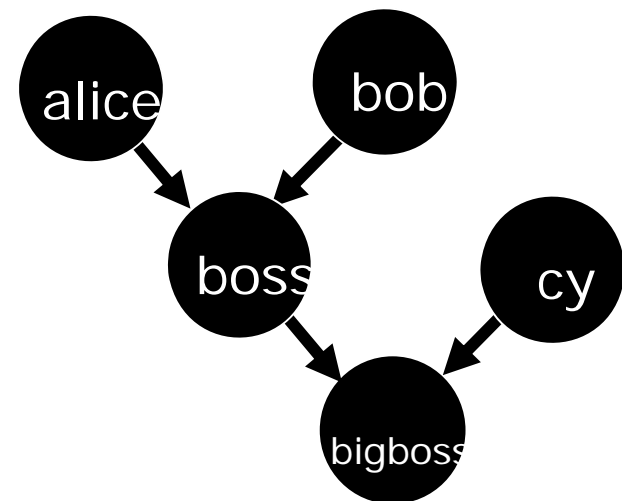
alice, bob, i cy
se mogu izvršavati paralelno

graf zavisnosti zadataka



Functional Level Parallelism sa sections direktivom

```
#pragma omp parallel
{
#pragma omp sections
{
#pragma omp section /* Optional */
    a = alice();
#pragma omp section
    b = bob();
#pragma omp section
    c = cy();
}
}
s = boss(a, b);
printf ("%6.2f\n",
        bigboss(s,c));
```



Primer2

```
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        (void) funcA();

        #pragma omp section
        (void) funcB();
    } /*-- End of sections block --*/

} /*-- End of parallel region --*/
```

Ako je funkcija funcA (slično i funcB) oblika

```
void funcA()
{
    printf("In funcA: this section is executed by thread %d\n",
        omp_get_thread_num());
}
```

Izlaz iz programa bi mogao da bude oblika

```
In funcA: this section is executed by thread 0
In funcB: this section is executed by thread 1
```

odredbe uz sections

```
private(list)  
firstprivate(list)  
lastprivate(list)  
reduction(operator:list) (C/C++)  
reduction({operator | intrinsic_procedure_name}:list) (Fortran)  
nowait
```

Single direktiva

- * Ovom direktivom se postiže da se strukturni blok izvršava od strane samo jedne niti.
 - Ona ne kaže koja je to nit, već samo da se taj blok izvršava sekvencijalno.
 - Druge niti čekaju na barijeri da se okonča izvršenje bloka koji se nalazi u okviru single direktive
 - Sintaksa

```
#pragma omp single [clause[ [, ] clause]... ] structured block
```

Primer

- * U sledećem primeru se koristi single direktiva da inicijalizuje deljivu promenljivu *a* :

```
#pragma omp parallel shared(a,b) private(i)
{
    #pragma omp single
    {
        a = 10;
        printf("Single construct executed by thread %d\n",
               omp_get_thread_num());
    }
    /* A barrier is automatically inserted here */

    #pragma omp for
    for (i=0; i<n; i++)
        b[i] = a;
} /*-- End of parallel region --*/

printf("After the parallel region:\n");
for (i=0; i<n; i++)
    printf("b[%d] = %d\n",i,b[i]);
```

- Implicitna barijera na kraju single konstrukcije veoma je važna.

Mogući izlaz iz programa
za n=9 i 4 niti

- Bez barijere, neke niti bi mogle da krenu sa

```
Single construct executed by thread 3
After the parallel region:
b[0] = 10
b[1] = 10
b[2] = 10
b[3] = 10
b[4] = 10
b[5] = 10
b[6] = 10
b[7] = 10
b[8] = 10
```


* Klauzule (odredbe) koje se mogu koristiti uz single direktivu su

```
private(list)  
firstprivate(list)  
copyprivate(list)  
nowait
```

Kombinovanje parallel direktive i direktiva za podelu posla

- * Moguće je kombinovati parallel direktivu i direktive za podelu posla da bi se skratilo pisanje koda, tj. direktivu kojom de definiše paralelni region sa *for* i *sections* direktivama
- * Npr. je ekvivalentno sa

```
#pragma omp parallel  
{  
    #pragma omp for  
    for (.....)  
}
```

```
#pragma omp parallel for  
for (.....)
```

kombinovanje direktiva

Full version	Combined construct
<pre>#pragma omp parallel { #pragma omp for for-loop }</pre>	<pre>#pragma omp parallel for for-loop</pre>
<pre>#pragma omp parallel { #pragma omp sections { [#pragma omp section] structured block [#pragma omp section structured block] ... } }</pre>	<pre>#pragma omp parallel sections { [#pragma omp section] structured block [#pragma omp section structured block] ... }</pre>

Oblast delovanja direktiva

- * FOR, SECTIONS i SINGLE direktive se moraju pojaviti unutar PARALLEL direktive
 - ali one ne moraju biti unutar iste funkcije

```
void main(void) {  
    dothework();  
    #pragma omp parallel  
    {  
        dothework();  
    }  
}
```

```
void dothework(void) {  
    #pragma omp for  
    for (ii=0; ii<N; ii++) {  
        ...  
    }  
}
```