

OpenMP

*Interakcija sa okruženjem

- Promenljive okruženja
- Runtime bibliotečke funkcije

*Odredbe – (nastavak)

interakcija sa radnim okruženjem

* Omp definiše interne *upravljачke promenljive* (*internal control variables*).

- Ovim promenljivim upravlja OMP implementacija.
- Ovim promenljivim se ne može direktno pristupati niti se mogu modifikovati na aplikativnom nivou;
 - ove promenljive se mogu testirati i modifikovati preko omp funkcija i promenljivih okruženje.

* Definisane su sledeće interne upravljачke promenljive:

- *nthreads-var* – pamti broj niti za izvršenje paralelnog regiona
- *dyn-var* – određuje da li će se dozvoliti dinamičko prilagođavanje broja niti kod izvršenja paralelnog regiona
- *nest-var* – definiše da li je dozvoljen ugnježdjeni paralelizam
- *def-sched-var* - pamti implementaciono definisani način distribucije iteracija petlje po nitima

- * Da bi se predefinisale vrednosti internih upravljačkih promenljivih na raspolaganju su četiri promenljive okruženja definisane standardom koje se mogu postaviti pre izvršenja programa
- * Za ovu svrhu mogu se koristiti i bibliotečke funkcije.
 - one predefinišu vrednosti postavljene preko promenljivih okruženja.
 - Da bi ove funkcije mogle da se koriste u C/C++ programima neophodno je uključiti omp.h header fajl.

Promenljive okruženja

* OMP_NUM_THREADS *integer*

- utiče na *nthreads-var* upravljačku promenljivu
 - npr. `setenv OMP_NUM_THREADS 8`

* OMP_DYNAMIC *flag*

- utiče na upravljačku promenljive *dyn-var*
 - flag može imati vrednost `true` ili `false`
- pomoću ove promenljive se može dozvoliti ili onemogućiti sistemu da dinamički prilagođava broj niti koje će se koristiti za izvršenje paralelnih regiona.
 - npr. `setenv OMP_DYNAMIC TRUE`

* OPM_NEST *flag*

- utiče na promenljivu *nest-var* i omogućava da se dozvoli ili onemogući korišćenje ugnježđenog paralelizma
 - npr. `setenv OMP_NESTED TRUE`
 - standard specificira da je po definiciji ova promenljiva inicijalizovana na *false*
 - Ako implementacija ne podržava ugnježđeni paralelizam, modifikacija interne upravljačke promenljive *nest-var* nema efekta.

* OMP_SCHEDULE

- utiče na upravljačku promenljivu *def-sched-var* koja definiše podrazumevani način distribucije iteracija u paralelnim for petljama
 - Npr. `setenv OMP_SCHEDULE "guided, 4"`
 - `setenv OMP_SCHEDULE "dynamic"`

OpenMP bibliotečke funkcije

* Funkcije koje se odnose na broj niti

- `void omp_set_num_threads (int) –`
 - postavlja broj niti u timu koje će se koristiti u okviru budućeg paralelnog regiona.
 - predefiniše broj niti postavljen promenljivom okruženja OMP_NUM_THREADS
- `int omp_get_num_threads (void) –`
 - vraća broj niti koji se koristi u tekućem paralelnom regionu.
- `int omp_get_max_threads (void)`
 - vraća koliko maksimalno niti može biti u paralelnom regionu.
- `int omp_get_thread_num (void)`
 - vraća redni broj (identifikator) niti u timu

OpenMP bibliotečke funkcije

* `int omp_get_num_procs (void)`

- vraća broj procesora koji je na raspolaganju programu

* `void omp_set_dynamic (int)`

- omogućava dinamičko prilagođavanje broja niti kod izvršenja budućeg paralelnog regiona

* `int omp_get_dynamic (void)`

- vraća TRUE ako je dozvoljeno dinamičko podešavanje broja niti, FALSE u protivnom

* `void omp_set_schedule (kind, chunksize)`

- predefiniše vrednost OMP_SCHEDULE promenljive okruženja

* `int omp_in_parallel (void)`

- vraća TRUE ako se pozove u okviru paralelnog regiona, u protivnom FALSE

* `void omp_set_nested (int)`

- dozvoljava ili zabranjuje ugnježđeni paralelizam
 - predefiniše vrednost promenljive okruženja OPM_NEST

* `int omp_get_nested (void)`

- vraća TRUE ako je dozvoljen ugnježđeni paralelizam, u protivnom FALSE

OpenMP bibliotečke funkcije

- `omp_get_wtime()`

- vraća vreme u sec od nekog trenutka u prošlosti

- `omp_get_wtick()`

- vraća kolika je preciznost časovnika koji se koristi u funkciji `omp_get_wtime()`.

Runtime Library routines

- To use a known, fixed number of threads in a program, (1) tell the system that you don't want dynamic adjustment of the number of threads, (2) set the number of threads, then (3) save the number you got.

```
#include <omp.h>
void main()
{  int num_threads;
   omp_set_dynamic( 0 );
   omp_set_num_threads( omp_num_procs() );
#pragma omp parallel
{  int id=omp_get_thread_num();
#pragma omp single
   num_threads = omp_get_num_threads();
   do_lots_of_stuff(id);
}
}
```

Disable dynamic adjustment of the number of threads.

Request as many threads as you have processors.

Protect this op since Memory stores are not atomic

Even in this case, the system may give you fewer threads than requested. If the precise # of threads matters, test for it and respond accordingly.

Još neke odredbe

- * if
- * num_threads
- * reduction
- * copyin

Odredba if

* odredba

```
if (skalarni_izraz)
```

* određuje da li će parallel direktiva dovesti do kreiranja tima niti.

- Samo jedna if odredba može da se nađe u okviru parallel direktive
- ako je vrednost logičkog izraza false, onda se paralelni region izvršava samo od strane jedne niti (tj. sekvencijalno)

Primer

```
#pragma omp parallel if (n > 5) default(none) \
    private(TID) shared(n)
{
    TID = omp_get_thread_num();
    #pragma omp single
    {
        printf("Value of n = %d\n",n);
        printf("Number of threads in parallel region: %d\n",
            omp_get_num_threads());
    }
    printf("Print statement executed by thread %d\n",TID);
} /*-- End of parallel region --*/
```

Izlaz iz programa za n=5 i n=10

- funkcija `omp_get_num_threads()`
- funkcija `omp_get_thread_num()` v
- Iskorišćena je direktiva `#pragma omp` naredbe štampanja više puta.

```
Value of n = 5
Number of threads in parallel region: 1
Print statement executed by thread 0
Value of n = 10
Number of threads in parallel region: 4
Print statement executed by thread 0
Print statement executed by thread 3
Print statement executed by thread 2
Print statement executed by thread 1
```

Odredba num_threads

- * Omogućava da definišemo koliko niti će biti kreirano pri ulasku u paralelni region
- * Sintaksa
 - `num_threads (celobrojni_pozitivan_izraz)`
- * Vrednost određena ovom odredbom redefiniše vrednost postavljenu pozivom bibliotečke funkcije `omp_set_num_threads()`,
 - koja pak redefiniše vrednost broja niti postavljenu korišćenjem promenljive okruženja, koja pak redefiniše default broj određen implementacijom
- * U najvećem broju slučajeva se koristiti vrednost koja je određena implementacijom
- * Koristi se samo uz direktivu `parallel!`

Kako definisati koliki broj niti će biti u paralelnom regionu

* Postoji više načinia

* Po prioritetu:

- – IF odredba
- – NUM_THREADS odredba
- – omp_set_num_threads() bibliotečka funkcija
- – OMP_NUM_THREADS promenljiva okruženja
- – Default: zavisno od implementacije

Primer

```
(void) omp_set_num_threads(4);
#pragma omp parallel if (n > 5) num_threads(n) default(none)\
    private(TID) shared(n)
{
    TID = omp_get_thread_num();
    #pragma omp single
    {
        printf("Value of n = %d\n",n);
        printf("Number of threads in parallel region: %d\n",
            omp_get_num_threads());
    }
    printf("Print statement executed by thread %d\n",TID);
} /*-- End of parallel region --*/
```

- primer ilustruje korišćenje odredbe *num_threads* i odredbe *if*
- Da bi ilustrovali prioritete pravila, ubačen je i poziv funkcije *omp_set_num_threads* kojom se broj niti postavlja na 4.
 - Ovo se odredbom *num_threads* predefiniše.

Odredba reduction

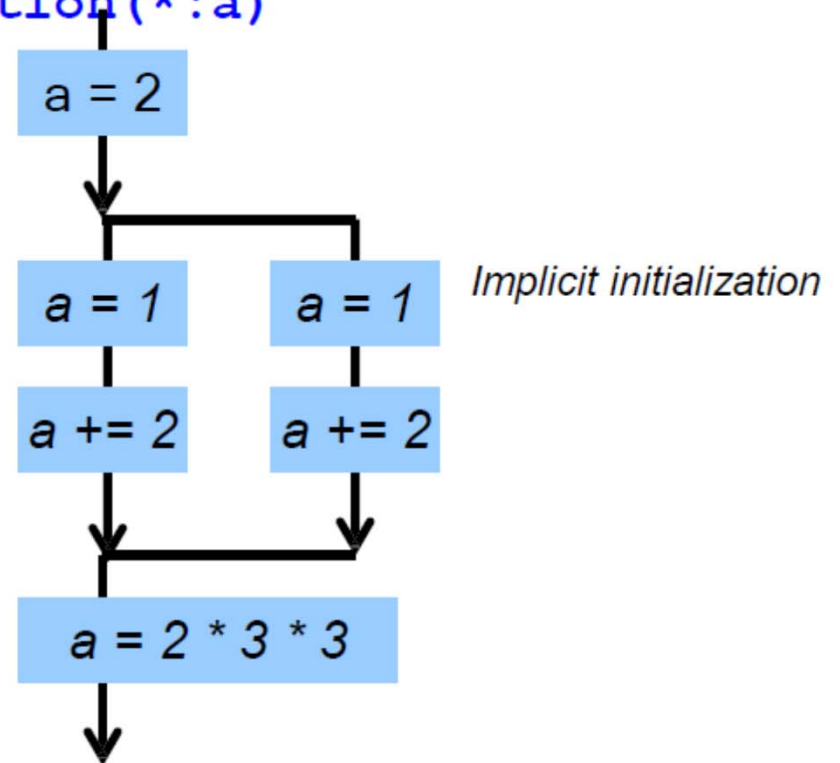
* Sintaksa

- `reduction(operator:lista_promenljivih)`
- Operetor može biti `+, -, *, &&, ||, &, |, ^`

- * Omogućava da se specificiraju neki oblici rekurentnog izračunavanja (u kojima se koriste komutativne i asocijativne operacije) tako da se ona mogu obaviti paralelno bez modifikacije koda.
- * Za svaku promenljivu koja se nalazi u listi promenljivih, kreira se privatna kopija sa inicijalnom vrednošću koju definiše operator redukcije (npr. ako je u pitanju operator `+` sve privatne kopije se inicijalizuju na 0, ako je u pitanju operator `*` sve privatne promenljive se inicijalizuju na 1, itd.)
- * Na kraju regiona na koji se odnosi redukcija, promenljiva na koju se odnosi redukcionni operator se ažurira tako što se originalna (polazna vrednost) promenljive kombinuje sa konačnim vrednostima svake privatne kopije.

Primer dejstva odredbe reduction

```
void main()  
{  
    int a = 2;  
    omp_set_num_threads(2);  
    #pragma omp parallel reduction(*:a)  
    {  
        a += 2;  
    }  
    printf("%d\n", a);  
}
```



Primer sumiranje elemenata polja

```
sum = 0;
for (i=0; i<n; i++)           sekvencijalno
    sum += a[i];
```

```
sum = 0;
#pragma omp parallel shared(n,a,sum) private(TID,sumLocal)
{
    TID = omp_get_thread_num();
    sumLocal = 0;
    #pragma omp for
        for (i=0; i<n; i++)
            sumLocal += a[i];
    #pragma omp critical (update_sum)
    {
        sum += sumLocal;
        printf("TID=%d: sumLocal=%d sum = %d\n",TID,sumLocal,sum);
    }
} /*-- End of parallel region --*/
printf("Value of sum after parallel region: %d\n",sum);
```

Paralelni kod bez korišćenja
reduction odredbe .
Zahteva korišćenje direktive
critical da bi se ažurirala
deljiva promenljiva *sum*

Sumiranje korišćenjem odredbe reduction

```
#pragma omp parallel for default(none) shared(n,a) \  
    reduction(+:sum)  
    for (i=0; i<n; i++)  
        sum += a[i];  
    /*-- End of parallel reduction --*/  
printf("Value of sum after parallel region: %d\n",sum);
```

moгуći operatori i inicijalne vrednosti promenljive koja je predmet redukcije

Operator	Initialization value
+	0
*	1
-	0
&	~0
	0
^	0
&&	1
	0

Promenljiva sum je privatana i inicijalizovana na 0 i deljiva promenljiva kojoj se pristupa uzajamno isključivo

NAD PROMENLJIVOM SUM NE SMEJU SE OBAVLJATI DRUGE OPERACIJE IZUZEV ONE KOJA JE U ODREDBI reduction navedena!

Threadprivate direktiva

* Sintaksa

- `#pragma omp threadprivate (lista)`

- * Ova direktiva omogućava da promenljive navedene u listi budu lokalne za svaku nit i da se prostiru kroz više paralelnih regiona.
- * Ova direktiva mora se naći nakon deklaracije promenljivih koje su navedene u listi i pre obraćanja tim promenljivim
- * Na ulasku u prvi paralelni region, promenljive navedene u listi mogu biti nedefinisane ako se ne koristi COPYIN naredba uz direktivu parallel.
- * Vrednosti koji su zapamćene u threadprivate promenljivim mogu se prostirati kroz više paralelnih regiona samo ako je broj niti konstantan.
 - Zbog toga obavezno treba isključiti dinamičko kreiranje niti pozivom funkcije
 - `omp_set_dynamic(0)`

Primer

```
#include <omp.h>
int a, b, i, tid; float x;
#pragma omp threadprivate(a, x)
main () {
/* Explicitly turn off dynamic threads */
  omp_set_dynamic(0);
  printf("1st Parallel Region:\n");
  #pragma omp parallel private(b,tid)
  {
    tid = omp_get_thread_num();
    a = tid;
    b = tid;
    x = 1.1 * tid + 1.0;
    printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x);
  } /* end of parallel section */

  printf("*****\n");
  printf("Master thread doing serial work here\n");
  printf("*****\n");
  printf("2nd Parallel Region:\n");
  #pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num();
    printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x);
  } /* end of parallel section */
}
```

Output:

1st Parallel Region:

Thread 0: a,b,x= 0 0 1.000000

Thread 2: a,b,x= 2 2 3.200000

Thread 3: a,b,x= 3 3 4.300000

Thread 1: a,b,x= 1 1 2.100000

***** Master thread doing serial
work here

***** 2nd Parallel Region:

Thread 0: a,b,x= 0 0 1.000000

Thread 3: a,b,x= 3 0 4.300000

Thread 1: a,b,x= 1 0 2.100000

Thread 2: a,b,x= 2 0 3.200000

Copyin odredba

- * Omogućava da se inicijalizuju vrednosti promenljivih navedenih u *threadprivate* direktivi na vrednosti tih promenljivih iz master niti (vrednosti tih promenljivih pre ulaska u prvi paralelni region)
- * Sintaksa
 - copyin (*lista_promenljivih*)
 - Promenljive koje se nalaze u *listi_promenljivih* moraju se prethodno pojaviti u odredbi *threadprivate*.
 - Ova odredba se može koristiti uz direktive *parallel*, *for* i *sections*

Task direktiva - eksplicitni task paralelizam

* Omogućava paralelizaciju iregularnih problema

- Beskonačne i while petlje
- Rekurzivne algoritme
- Producer/consumer problem

* Task predstavlja nezavisnu jedinicu izvršenja

- Task se sastoji od
 - Koda koji treba izvršiti
 - Podataka (deljivih i privatnih)
- Niti izvršavaju različite taskove
 - task može izvršiti odmah nakon kreiranja, ili
 - Izvršenje taska može biti odloženo

Task direktiva

#pragma omp task [*clause*[[,*clause*] ...]
structured-block

where *clause* can be one of:

if (*expression*) – ako je vrednost *expression*=*false*,
task će odmah krenuti sa izvršenjem

untied

shared (*list*)

private (*list*)

firstprivate (*list*)

default(shared | none)

Tied & Untied Tasks

- Tied Tasks:
 - tied task (vezani zadatak) kada se kreira dodeljuje mu se nit koja će ga izvršavati tokom celog životnog veka taska
 - Podrazumevano je (default) da je task tied (tj. vezan)
- Untied Tasks:
 - untied task (nevezani zadatak) mogu izvršavati različite niti tokom životnog veka zadatka.
 - untied task se kreira korišćenjem odredbe "untied"
 - Primer: `#pragma omp task untied`

Task direktiva – primer

- Tim niti se kreira na ulasku u omp parallel direktivu
- Jedna nit se odabira da izvrši while petlju, recimo nit "L"
- Nit L izvršava while petlju, kreira taskove, i pribavlja sledeće pointere u lančanoj listi.
- Svaki put kada nit L naiđe na omp task direktivu, ona kreira novi task i dodeljuje mu nit.
- Svaki task se izvršava u okviru svoje niti.
- Svi taskovi se moraju sinhronizovati na barijeri na kraju single direktive.

```
#pragma omp parallel
{
    #pragma omp single
    { // block 1
        node * p = head;
        while (p) { //block 2
            #pragma omp task
            process(p);
            p = p->next; //block 3
        }
    }
}
```

Mogući scenario izvršenja

Korišćenjem taskova mogu se paralelizovati while petlje i rekurzivni programi

```
#pragma omp parallel
{
  #pragma omp single
  { // block 1
    node * p = head;
    while (p) { //block 2
      #pragma omp task
      process(p);
      p = p->next; //block 3
    }
  }
}
```

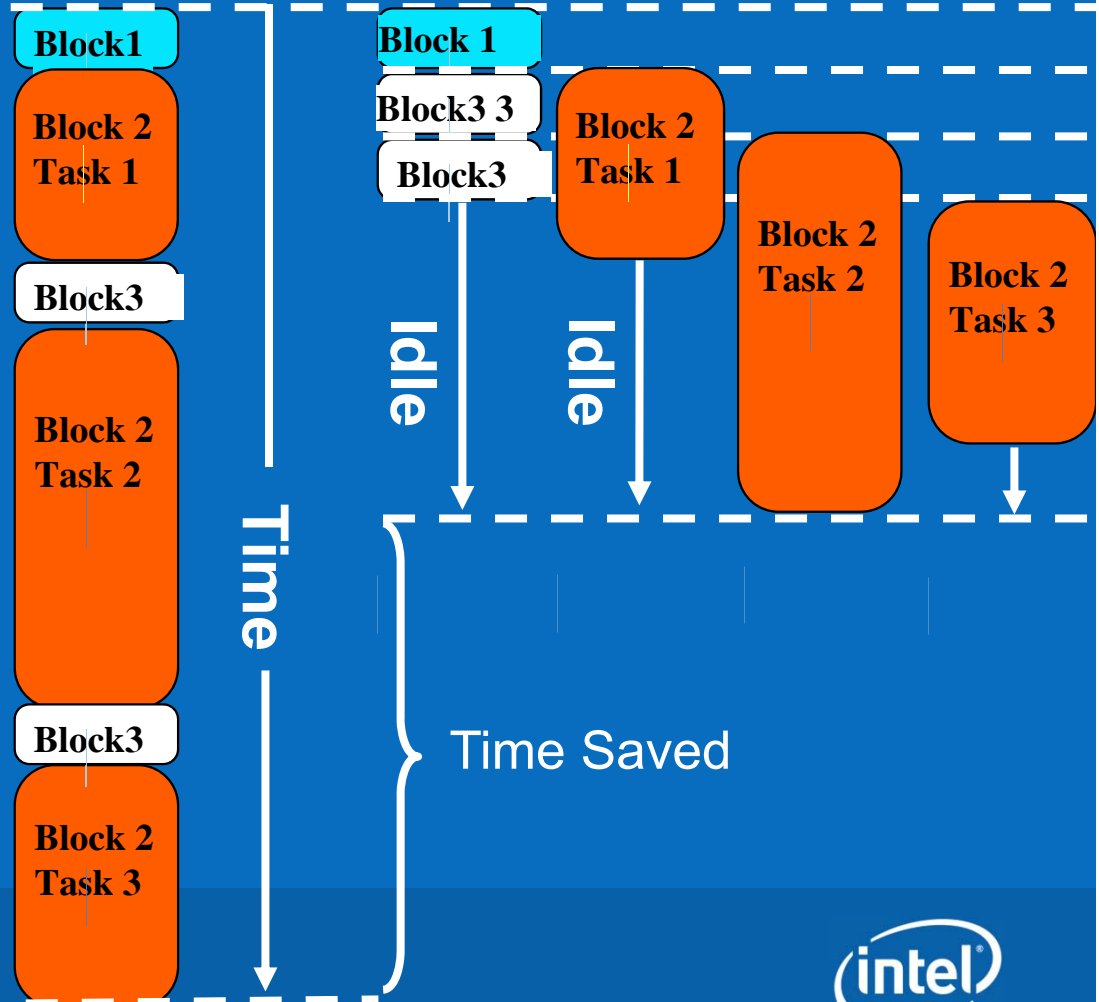
Single
Threaded

Thr1

Thr2

Thr3

Thr4



Gde se taskovi sinhronizuju (okončavaju izvršenje)?

- Na implicitnim barijerama (na kraju paralelnog regiona, paralelne for petlje, single direktive)
- Na eksplicitnim barijerama, korišćenjem direktive: `#pragma omp barrier`
- Korišćenjem `#pragma omp taskwait`

Primer okončanja zadataka

```
#pragma omp parallel
{
    #pragma omp task
    foo();
    #pragma omp barrier
    #pragma omp single
    {
        #pragma omp task
        bar();
    }
}
```

Multiple foo tasks created here – one for each thread

All foo tasks guaranteed to be completed here

One bar task created here

bar task guaranteed to be completed here

Zadatak

*Napisati program koji štampa "A race car" ili
"A car race" i maksimizira paralelizam*

Primer 1/1

sekvencijalni kod

```
#include <stdlib.h>
#include <stdio.h>
```

A race car

```
int main(int argc, char *argv[])
{

printf("A ");
printf("race ");
printf("car ");

printf("\n"); return(0);
}
```

Šta će ovaj program odštampati ?

Primer 1/2

```
#include <stdlib.h>  #include
<stdio.h>

int    main(int      argc, char

*argv[]) {

#pragma omp parallel
{
printf("A ");
printf("race ");
printf("car ");
} // End of parallel region

    printf("\n");
    return(0);
}
```

***Šta će program odštampati
ako je broj niti 2 ?***

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car A race car
```

"A A race race car car" ili

"A race A car race car", ili

"A race A race car car", ili

.....

Primer 1/3

```
#include <stdlib.h> #include <stdio.h>
int main(int argc, char *argv[])
{
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("A ");
            printf("race ");
            printf("car ");
        }
        // End of parallel region

        printf("\n"); return(0);
    }
}
```

***Šta će ovaj program odštampati
ako se kreiraju 2 niti ?***

Primer 1/3

```
$ cc -xopenmp -fast hello.c  
$ export OMP_NUM_THREADS=2  
$ ./a.out  A race car
```

***Ali samo jedna nit obavlja
posao.....***

Primer 1/4

```
int main(int argc, char *argv[])
{
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("A ");
            #pragma omp task
            {printf("race ");}
            #pragma omp task
            {printf("car ");}
        }
    } // End of parallel region
    printf("\n");
    return(0);
}
```

***Šta će ovaj program odštampati
ako se kreiraju 2 niti ?***

Primer 1/5

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out  A race car
$ ./a.out  A race car
$ ./a.out  A car race
$
```

***Zadaci se mogu izvršavati u
proizvoljnom redosledu***

Primer 2/1

hoćemo da se rečenica završava sa “is fan to watch”

```
int main(int argc, char *argv[]) {  
    #pragma omp parallel  
    {  
        #pragma omp single  
        {  
            printf("A ");  
            #pragma omp task  
            {printf("race ");}  
            #pragma omp task  
            {printf("car ");}  
            printf("is fun to watch ");  
        }  
        // End of parallel region  
  
        printf("\n");  
        return(0);  
    }  
}
```

***Šta će ovaj program odštampati
ako se kreiraju 2 niti ?***

```
$ cc -xopenmp -fast hello.c  
$ export OMP_NUM_THREADS=2  
$ ./a.out
```

```
A is fun to watch race car  
$ ./a.out
```

```
A is fun to watch race car  
$ ./a.out
```

```
A is fun to watch car race  
$
```

```
#pragma omp parallel
{
    #pragma omp single
    {
        printf("A ");
        #pragma omp task
        {printf("car ");}
        #pragma omp task
        {printf("race ");}
        #pragma omp taskwait
        printf("is fun to watch ");
    }
} // End of parallel region

printf("\n");return(0);
}
```

***Šta će ovaj program odštampati
ako se kreiraju 2 niti ?***

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
$
A car race is fun to watch
$ ./a.out
A car race is fun to watch
$ ./a.out
A race car is fun to watch
$
```

Prvo se izvršavaju zadaci jer smo postavili taskwait direktivu