When END PARALLEL

(parallelism initiated)

whether or not the threads are newly created or simply
receive and via other parallelism specifications.

When PARALLEL

parallelism initiator.

The PARALLEL/END PARALLEL pair is the main
complex parallel construction capability.

constructs are in fact specifications of a more

The PARALLEL DO and PARALLEL SECTIONS

team is disbanded and sequential execution resumes.
To suppress sharing the work and to synchronize the work, started to share work specified in the code each receives, 

There are several constructs that allow the threads

parallelism.

... sequentialized. But the standard allows nested team of one thread, i.e., the lower levels are parallelism. Currently, the second level is run with a nested, a second team is formed to support nested constructs are...
i$omp END DO

end do
call work(j',x',n')
    do j=1',n'

i$omp DO SCHEDULE(RUNTIME), PRIVATE(j')

end do
call local(jam',x',n',w')
    do i=1',w'

jam = OMP_GET_NUM_THREADS()

i$omp PARALLEL DEFAULT(SHARED), PRIVATE(j',jam')
OMP END SINGLE
  call input(\textasciitilde'\textasciitilde,n)
call output(x,\textasciitilde'\textasciitilde,n)
OMP SINGLE
OMP END SECTIONS

OMP SINGLE

OMP END SECTIONS

OMP SINGLE

OMP END SECTIONS

OMP SINGLE

OMP END SECTIONS

OMP SINGLE
otherwise:

may enter at different times unless explicitly told

DO/END DO. All threads must encounter it but they

There is no implied barrier at the beginning of the

carlier.

shared by all threads. It has all of the clauses discussed

The DO/END DO pair specifies a loop whose work is

each thread runs all m iterations on local data.

The first do loop runs sequentially on each thread, i.e.,

PARALLEL/END PARALLEL pair.

sends each a copy of the code specified between the

The PARALLEL construct starts the threads and
a single DO/ENDDO

the same as a PARALLEL END PARALLEL

The PARALLEL DO/END PARALLEL DO

• reached the END DO unless explicitly told otherwise.

• Threads will wait there until all threads have DO. There is an implied barrier at the end of the DO/ENDDO.
otherwise.

unless explicitly stated

PARALLEL SECTIONS

No barrier is implied at the beginning of the

concurrent.

SECTIONS / END PARALLEL SECTIONS

The clauses are as before with the PARALLEL

sections that are each run by a (possibly distinct) thread.

The SECTIONS / END SECTIONS

•
SECTIONS

PARALLEL with a single SECTIONS/END
PARALLEL pair is the same as a PARALLEL/END SECTIONS pair unless explicitly stated otherwise.

A barrier is implied at the END PARALLEL
unless explicitly stated otherwise.

- No barrier is implied at the beginning of the SINGLE

- Explicitly test the thread number.

- You do not know which thread runs this code unless you explicitly test the thread number.

- Routine: Here it is used to do I/O assuming reentrant I/O

- Code that only one thread from the team executes.

- The SINGLE/END SINGLE pair defines a section of
threads.

the barrier at the END SINGLE and wait for all other

Those that do not execute the code go immediately to

must reach this section.

As with the other worksharing constructs, all threads

explicitly stated otherwise.

A barrier is implied at the END SINGLE unless
within a parallel region in the same order.

• All threads must encounter all worksharing directives.

• The END PARALLEL has an implied barrier.

• The END PARALLEL routine.

In this case, all threads then execute a call to the work

• PARALLEL construct.

threads continue with the code sent to each by the

• After the END SINGLE barrier in the example the

parallel is more restrictive than a critical section.

• Note that the function of the SINGLE/END SINGLE
completed their part of the first loop before the others.
sharing on the second can begin for threads who have
Suppose we have two independent parallel loops. Work
This can be overcome by a NO WAIT clause.
at the end directive.
Note that most of these constructs have implied barriers
iomp end parallel
iomp end do
end do

z(j) = c(j)
do j=1,n
iomp do schedule(runtime), private(f)
end do

q(j) = a(j)
do j=1,n
iomp do schedule(runtime), private(f)
end do

iomp parallel default(shared)
barrier is implied.

master executes the block of code. No entry or exit

The MASTER/END MASTER pair implies only the

particular thread to execute the code in the block.

A slightly modified version allows the selection of a

form of synchronization.

The SINGLE/END SINGLE pair is a quite restrictive
barrier on exit.

work sharing constructs that only have an implied

It is also useful to enforce a barrier at the entry to the

the team executing a parallel region.

This can be used to enforce a barrier across all threads in

The simplest is the **Barrier**

(Events)

synchronization mechanisms we have discussed. (Except

Open MP supports a form of almost all of the
 !$OMP PARALLEL DEFAULT(SHARED), PRIVATE(i,iam)
  iam = OMP_GET_NUM_THREAD()
  do i=1,m
       call local(iam,x,n,m)
  end do
 !$OMP BARRIER
 !$OMP DO SCHEDULE(RUNTIME), PRIVATE(j)
  do j=1,n
       call work(j,x,n)
  end do
 !$OMP END DO
 !$OMP END PARALLEL
Notice the name associated with each.

The following has two critical sections.

Only one thread is allowed to enter: the SINGLE construct which is a critical section that all threads may pass through the section as opposed to thread at any given time.

The code in the section can only be executed by a single thread.

This can be used in any parallel setting.

There is also support for a critical section.
$omp parallel
$i
omp end parallel

$omp end critical (ysum) 

$omp parallel 
$i
omp critical (ysum) 

$omp end critical (ysum) 

$omp parallel 
$i
omp critical (xsum) 

$omp end critical (xsum) 

$omp parallel 

$omp critical 

$omp end critical 

$omp parallel default (shared), private (i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, [, ], ^, \) 

$omp end parallel
protected in the following example.

It only applies to the next statement so simply not be used.

be atomic in its application the \texttt{ATOMIC} directive can be atomic in its application the \texttt{ATOMIC} directive can

If the critical section is simply an update that needs to

•
i$omp END PARALLEL DO
end do

totally = totaly + sumy

sumx = call local(1)'x',n)

sumx = call local(1)'x',n)
do i = 1,'n
i$omp PARALLEL DO
will not reach it).

Lower iterations have completed it (or it is known they
Each thread enters the ordered segment only after all
reduction among other things.
It is useful to guarantee the order of summation in a
appear both lexically and dynamically in a loop iteration.
only one ordered/END ordered pair can
ordered/END ordered pair
delimited by a ordered/END ordered pair.
parallel do loop it may contain a code segment
If an ordered clause is used in a DO or
AWAIT can be used.
In the case of a parallel do loop a version of ADVANCE
$\text{do } i = 1, n$

$\text{sum}_{x} = \text{call local}(i, x, n)$

$\text{sum}_{y} = \text{call local}(i, y, n)$

$\text{total}_{x} = \text{total}_{x} + \text{sum}_{x}$

$\text{total}_{y} = \text{total}_{y} + \text{sum}_{y}$

$\text{end do}$

$\text{end } \$OMP$\text{ PARALLEL}$

$\text{do}$

$\text{end } \$OMP$\text{ END PARALLEL}$
!omp parallel do
end do

totx = totx + sumx
toty = tota+y + sumy

!OMP PARALLEL DO ORDERED, SCHEDULE(DYNAMIC),
!OMP END PARALLEL DO
end do

sumx = call local (1, x,n)
sumy = call local (1, y,n)

do i = 1,n

!OMP PARALLEL REDUCTION(+: tota+y, tota+y)

It can be applied to associative operators.

It is also possible to allow a reduction operation but not
OMP DESTRUCTOR

A lock variable can be liberated from use as a lock via:

locked then it behaves as OMP-SET-LOCK.

OMP-TEST-LOCK allows the routine to return when
encountering a locked lock. If the lock was successfully
locked a locked lock will cause the thread to wait until
it is given the lock.

They can be locked and unlocked as shown.

OMP INIT-LOCK

Lock variables must be initialized explicitly. Using
man omp-lock contains the details.

OMP INIT-LOCK contains the details.

There is support for locks in order to allow a user to
implement synchronization primitives.
CALL OMP-UNSEML-LOOK(1CK)

PRINT *, 'MY THREAD ID IS ', ID

CALL OMP-SET-LOOK(1CK)

ID = OMP-GET-THREAD-NUM()

!$OMP PARALLEL SHARED(1CK) PRIVATE(ID)

CALL OMP-INIT-LOOK(1CK)

; ARE ASSUMED HERE

; SIZED 64 BIT ADDRESSES

INTEGER*8 LCK ; THIS VARIABLE SHOULD BE POINTER

LOGICAL OMP-TEST-LOOK

EXTERNAL OMP-TEST-LOOK

PROGRAM LOCK-USEAGE
CALL OMP_DESTROY_LOCK(ck)

END

CALL OMP_UNSET_LOCK(ck)

!$OMP END PARALLEL

CALL OMPiscard(ID) ! WE NOW HAVE THE LOCK

END DO

CALL OMPiscard(ID) ! WE DO NOT YET HAVE THE LOCK

DO WHILE (.NOT. OMP тест LOCK(ck))