provides multi-thread parallelism.
↑
OpenMP. (open specification for multi processing)
↓
- Starts with a single thread (master thread)
- Additional threads (team) are created when master hit a parallel region.

Example:-

```
void main()
{
double Res[1000];
for(int i=0; i<1000; i++)
  { do_huge_comp(Res[i]);
  }
}
```
Sequential Program

```
void main ()
{ double Res [1000];
#pragma omp parallel for
for (int i=0; i<1000; i++) {
 do_huge_comp ( Res[i]);}
}
```
parallel program.

Openmp derivatives

```
#pragma omp parallel private (var)
{
int threadId=omp_get_thread_num();
}
```
↓
Each thread gets its own private copy of variable var.

omp_set_num_thread
omp_get_num_thread

#include <omp.h>

— In OPENMP, clauses are used to specify behaviour of parallel region.

a) ~~#~~ #pragma omp parallel if (a > 2)

b) #pragma omp parallel num_threads (8).

c) #pragma omp parallel private (var)

d) #pragma omp parallel firstprivate (var)

Har thread ko apni copy milti hy, jo master thread ki value se initilize hoti hy at start of ~~parallel region~~

↓ ~~Similar to private~~ but variables in the list are initilized with ~~values they had before~~ entering parallel region.

e) #pragma omp parallel shared (var1, var2)

all threads aces same memory location for shared variables

↓ Variables in the list are shared amoung all threads in parallel region.

f) #pragma omp parallel default (none | shared | private)

no variable is shared or private by default.

g) " copyin (var) → • threads created dynamically.
• copy values of variables from master thread into threads that are created.

h) #pragma omp parallel for reduction (+ : sum).

operator

list.

Each thread gets private copy of variable, & at end of parallel region private copies are combined using specific operator.

~~Each~~ has private copy of sum.

export ~~[illegible]~~ OMP_NUM_THREADS = 4.
echo // x

- default is share.

- Parallel region is a block of code that will be executed by multiple threads
- When (in serial program) a parallel derivative ~~[illegible]~~ team of thread is created. + main thread becomes master of the team. + master thread has id or number-0
- code duplicates ~~[illegible]~~ thread executes

- # pragma omp parallel if (np>1) num_threads(np)
  { }
- omp_set_num_threads (8);
- # pragma omp parallel
  { }

- int omp_in_parallel ( ) → if(omp_in_parallel ( ))
  { }.

  ↓
  returns zero if executing outside a parallel region
  returns non-zero if code is executing in parallel region

- # pragma omp parallel for lastprivate (list).
  ↓ used in for loops.

- Har thread ko apni alag copy ~~milti hai~~, lekin parallel region ke end mein, jo last thread execute karta hai usko value master thread ke variable mein copy ho jata hai.

(1) int list[5] = {1,2,3,4}...

```
• #pragma omp parallel default(private) shared(list)
  {
       int sum;

       ---

  }
```

list is shared but sum variable is private to each thread, each thread gets its own copy of sum.

Imp!!

(2)
```
        int x=10;
   #pragma omp parallel default(shared) private(x)
```

Imp!! x will be private to each thread.

(3)
```
   #pragma omp parallel for default(shared)
        for (i=0 ...)
            { b = a+i }
```

wrong as race condition

```
#pragma omp parallel for reduction(+ : b) shared(a)
        for ( - )
            { (b=a+i) }
```

variable jis par karna h

+, -, *, min, max

Master thread is 0

Synchronized (: wait for other thread)

•– Getting id of current thread

```
int a;
int b;
#pragma omp parallel private (a,b) num_threads (2)
{       a = omp_get_thread_num();   // ID of current thread
        b = omp_get_num_thread ();  // total threads that are
                                        executing parallel region

     if (a==0) { master thread }
```

•– # pragma omp parallel for schedule (static, 2)
"   " (dynamic, 2)

(static, 2) → divides loop into fixed size chunks & assign them to threads in advance. Best for uniform work (nem iterations take similar time).

(dynamic, 2) → work stealing

•– Synchronized
# pragma omp for

Non- Synchronize
# pragma omp for nowait

•– Guided Scheduling: gives chunks of work to thread but as work goes, chunk size decreases

» If no chunksize then it decreases to 1.

• – Ordered clause (comes only in loop)
Loop iterations are executed in same order
as they would be in sequential loop.

```
# pragma omp parallel for ordered
for (i=0; i<N; i++)
{ # pragma omp ordered
    cout << "Iteration" << i << endl;
}
```

[illegible]

• – #pragma omp critical (only 1 thread can execute a ~~thread~~ block at a time)
ensures mutual exclusion.

• – # pragma omp atomic
(makes only one specific operation atomic)
no 2nd thread can modify anything
x+=y;

• – #pragma omp master.
{ code can only be executed by master thread }

• – #pragma omp barrier (synchronize all threads within parallel region)

When thread reaches barrier, it waits until all other threads in parallel region also reach the barrier.