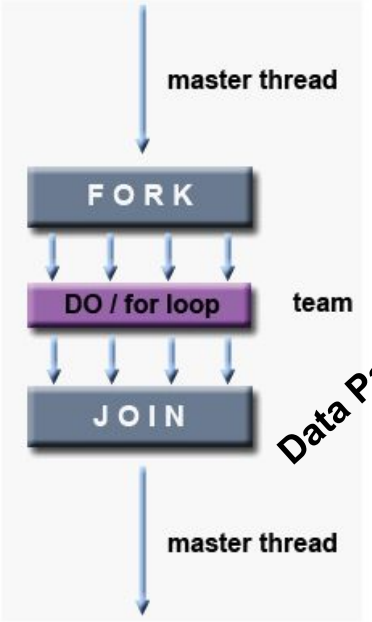
OpenMP:
Work Sharing Constructs

John H. Osorio Ríos

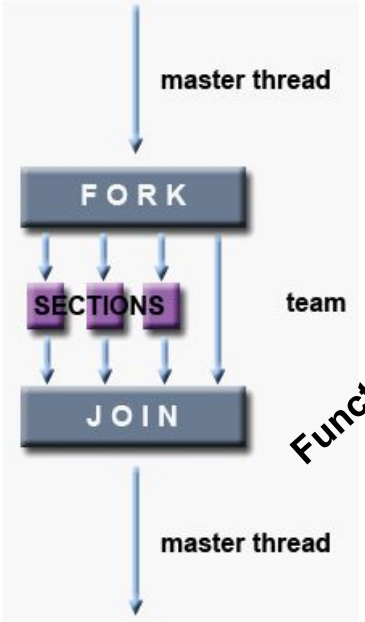
Work Sharing Constructs

- A work-sharing construct divides the execution of the enclosed code region among the members of the team that encounter it.
- Work-Sharing constructs do not launch new threads
- There is no implied barrier upon entry to a work-sharing construct, however there is an implied barrier at the end of a work sharing construct.

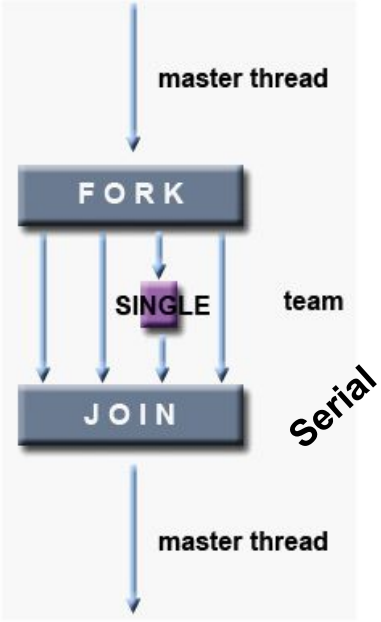
Types



Data Parallelism



Functional Parallelism



Serial

Restrictions

- A work-sharing construct must be enclosed dynamically within a parallel region in order for the directive to execute in parallel.
- Work-sharing constructs must be encountered by all members of a team or none at all.
- Successive work-sharing constructs must be encountered in the same order by all members of a team.

DO/FOR Directive

- The DO/for directive specifies that the iterations of the loop immediately following it must be executed in parallel by the team. This assumes a parallel region has already been initiated, otherwise it executes in serial on a single processor.

DO/FOR Directive

```
#pragma omp for [clause ...]  newline
                schedule (type [,chunk])
                ordered
                private (list)
                firstprivate (list)
                lastprivate (list)
                shared (list)
                reduction (operator: list)
                collapse (n)
                nowait

    for_loop
```

DO/FOR Clause

- **Schedule:** Describes how iterations of the loop are divided among the threads in the team. (read)
  - Static: Loop iterations are divided into pieces of size chunk and then statically assigned to threads. If chunk is not specified, the iterations are evenly (if possible) divided contiguously among the threads.

DO/FOR Clause

- Dynamic: Loop iterations are divided into pieces of size chunk, and dynamically scheduled among the threads; when a thread finishes one chunk, it is dynamically assigned another. The default chunk size is 1.

DO/FOR Clause

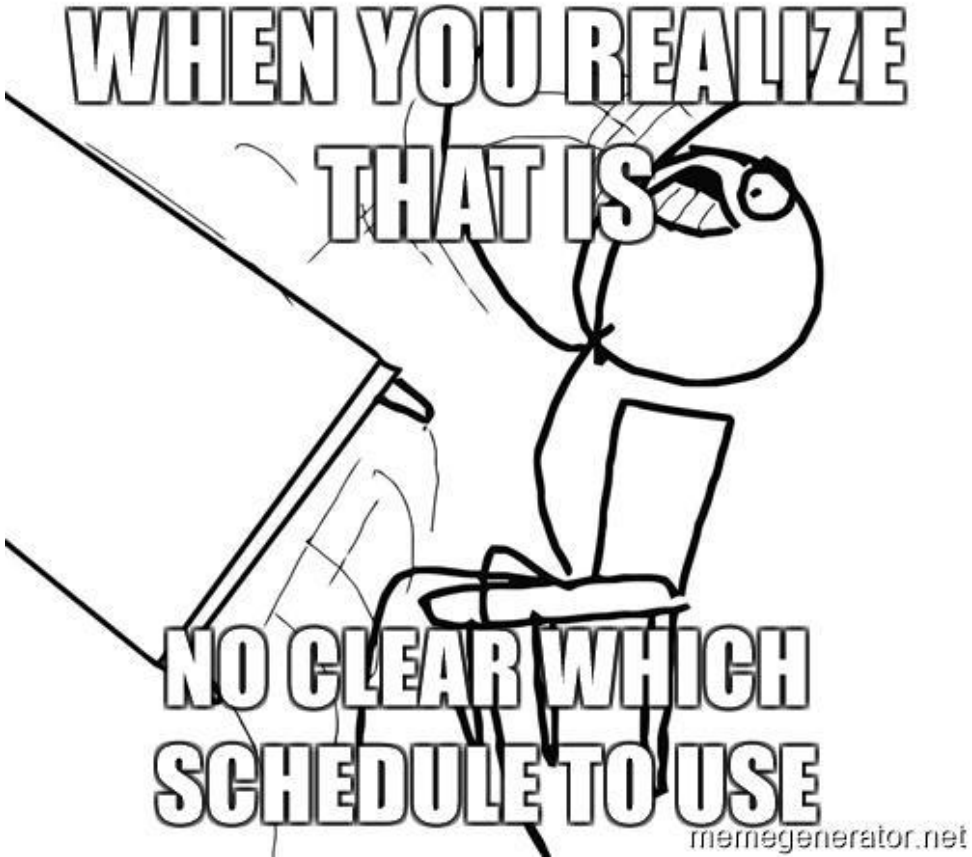
- Guided: Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned. Similar to DYNAMIC except that the block size decreases each time a parcel of work is given to a thread. The size of the initial block is proportional to:

  number_of_iterations / number_of_threads
  Subsequent blocks are proportional to
  number_of_iterations_remaining / number_of_threads
  The chunk parameter defines the minimum block size. The default chunk size is 1.

DO/FOR Clauses

- Runtime: The scheduling decision is deferred until runtime by the environment variable **OMP_SCHEDULE**. It is illegal to specify a chunk size for this clause.
- Auto: The scheduling decision is delegated to the compiler and/or runtime system.

DO/FOR Clauses



DO/FOR Clause

- **NO WAIT/nowait**: If specified, then threads do not synchronize at the end of the parallel loop.
- **ORDERED:** Specifies that the iterations of the loop must be executed as they would be in a serial program.

DO/FOR Clause

- **COLLAPSE**: Specifies how many loops in a nested loop should be collapsed into one large iteration space and divided according to the **schedule** clause. The sequential execution of the iterations in all associated loops determines the order of the iterations in the collapsed iteration space.

DO/FOR Clause Restrictions

- The DO loop can not be a DO WHILE loop, or a loop without loop control. Also, the loop iteration variable must be an integer and the loop control parameters must be the same for all threads.
- Program correctness must not depend upon which thread executes a particular iteration.

DO/FOR Clause Restrictions

- It is illegal to branch (goto) out of a loop associated with a DO/for directive.
- The chunk size must be specified as a loop invariant integer expression, as there is no synchronization during its evaluation by different threads.
- ORDERED, COLLAPSE and SCHEDULE clauses may appear once each.

Example: DO/FOR Directive

- Simple vector-add program
  - Arrays A, B, C, and variable N will be shared by all threads.
  - Variable I will be private to each thread; each thread will have its own unique copy.
  - The iterations of the loop will be distributed dynamically in CHUNK sized pieces.
  - Threads will not synchronize upon completing their individual pieces of work (NOWAIT).

Example: DO/FOR Directive

```
1  #include <omp.h>
2  #define N 1000
3  #define CHUNKSIZE 100
4
5  main(int argc, char *argv[]) {
6
7  int i, chunk;
8  float a[N], b[N], c[N];
9
10 /* Some initializations */
11 for (i=0; i < N; i++)
12   a[i] = b[i] = i * 1.0;
13 chunk = CHUNKSIZE;
14
15 #pragma omp parallel shared(a,b,c,chunk) private(i)
16   {
17
18   #pragma omp for schedule(dynamic,chunk) nowait
19   for (i=0; i < N; i++)
20     c[i] = a[i] + b[i];
21
22   }   /* end of parallel region */
23
24 }
```

THANKS

john@sirius.utp.edu.co