Parallel

Parallel is the clause that tells OpenMP to create a parallel region, in which threads are spawned. The number of threads spawned in parallel regions is specified by the environment variable OMP_NUM_THREADS. It can be overriden during program execution via the OpenMP routine omp_set_num_threads.

```
#pragma omp parallel [clause[[,]clause]...] <new-line>
   <structured-block>
```

Clauses:

The clause to appear on the parallel construct, which is one of the following:

- num_threads(integer-expression)
- private(list)
- hared(list)
- eduction([modifier,]identifier : list)

Example: https://rookiehpc.org/openmp/docs/parallel/index.html

Critical

The **critical** construct, which is used inside parallel regions, tells OpenMP that the associated block is to be executed by every thread but no more than one thread at a time. The critical construct must not be confused with the single or master constructs.

```
#pragma omp critical [(name) [[,] hint(hint-expression)]] <new-line>
   <structured-block>
```

Example: https://rookiehpc.org/openmp/docs/critical/index.html

Atomic

Atomic provides mutual exclusion but only applies to the update of a memory location (the update of X in the following example). The atomic directive can be applied only if a critical section consist of a single assignment statement that updates a scalar variable.

```
#pragma omp atomic <new-line>
   <expression-stmt>
```

Example: https://rookiehpc.org/openmp/docs/atomic/index.html

FOR

The **for** construct tells OpenMP that the iteration set of the for loop that follows is to be distributed across the threads present in the team. Without the for construct, the entire iteration set of the for loop concerned will be executed by each thread in the team. From a syntactic point of view, note that the for construct must be immediately followed by the for loop; it does not allow curly brackets between it and the for loop.

```
#pragma omp for [clause[[,]clause]...] <new-line>
   <for-loops>
```

Clauses:

The clause to appear on the for construct:

- private(list)
- reduction([modifier,]identifier : list)
- schedule([modifier[, modifier] :]kind[,chunk_size])
- ordered[(n)]
- nowait

Example: https://rookiehpc.org/openmp/docs/for/index.html

Reduction

The reduction clause indicates that the variables passed are, as its name suggests, used in a reduction. Each implicit task or SIMD lane creates a private copy initialises it to the initialiser value of the reduction identifier, that is, 0 for a sum or 1 for a product to name a few. After the end of the region, the original list item is updated with the values of the private copies using the combiner associated with the reduction identifier.

By default, the reduction computation is complete at the end of the construct. However, if nowait is specified on the construct, this is no longer guaranteed. Indeed, accesses to the original list item will create a data race and, thus, have unspecified effect unless synchronisation ensures that they occur after all threads have executed all of their iterations or section constructs, and the reduction computation has completed and stored the computed value of that list item. This can most simply be ensured through a barrier synchronisation.

```
#pragma omp parallel for reduction([modifier, ]identifier: list)
   <for-loops>
```

Example: https://rookiehpc.org/openmp/docs/reduction/index.html

Barrier

The barrier construct, which is a stand-alone directive, specifies an explicit synchronisation barrier at the point at which the construct appears. The barrier applies to the innermost enclosing parallel region, forcing every thread that belong to the team of that parallel region to complete any pending explicit task. Only once all threads of that team satisfy this criterion will they be allowed to continue their execution beyond the barrier.

```
#pragma omp barrier <new-line>
```

Example: https://rookiehpc.org/openmp/docs/barrier/index.html

Nowait

The **nowait** clause removes the implicit barrier that is present at the end of worksharing (sections, single, workshare) and target constructs.
If nowait is specified on a construct where a reduction clause is, accesses to the original list item after the end of the construct will create a data race and, thus, have unspecified effect unless synchronisation ensures that they occur after all threads have executed all of their iterations or section constructs, and the reduction computation has completed and stored the computed value of that list item. This can most simply be ensured through a barrier synchronisation.

```
#pragma omp for nowait <new-line>
```

Example: https://rookiehpc.org/openmp/docs/nowait/index.html

Master

Master is a clause that must be used in a parallel region; it tells OpenMP that the associated block must be executed by the master thread only. The other threads do not wait at the end of the associated block as if there was an implicit barrier. The master clause must not be confused with the single of critical clauses.

```
#pragma omp master <new-line>
   <structured-block>
```

Example: https://rookiehpc.org/openmp/docs/master/index.html

Single

Single is a clause that must be used in a parallel region; it tells OpenMP that the associated block must be executed by one thread only, albeit not specifying which one. The other threads will wait at the end of the associated block, at an implicit barrier, unless the single clause is accompanied with a nowait clause. The single clause must not be confused with the master or critical clause.

```
#pragma omp single [clause[[,]clause]...] <new-line>
   <structured-block>
```

Clauses:

The clause to appear on the single construct, which is one of the following:

- private(list)
- nowait

Example: https://rookiehpc.org/openmp/docs/single/index.html