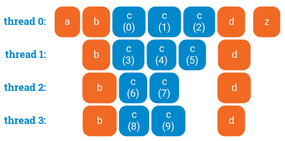
Chapter 3: Multithreading with OpenMP

OpenMP parallel for loops: waiting

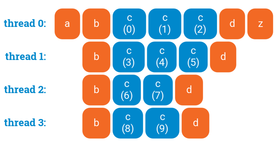
When you use a `parallel` region, OpenMP will automatically wait for all threads to finish before execution continues. There is also a synchronization point after each `omp for` loop; here no thread will execute `d()` until all threads are done with the loop:

```
a();
#pragma omp parallel
{
    b();
    #pragma omp for
    for (int i = 0; i < 10; ++i) {
        c(i);
    }
    d();
}
z();
```



However, if you do not need synchronization after the loop, you can disable it with `nowait`:

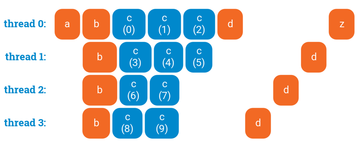
```
a();
#pragma omp parallel
{
    b();
    #pragma omp for nowait
    for (int i = 0; i < 10; ++i) {
        c(i);
    }
    d();
}
z();
```



Interaction with critical sections

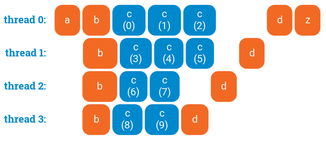
If you need a critical section after a loop, note that normally OpenMP will first wait for all threads to finish their loop iterations before letting any of the threads to enter a critical section:

```
a();
#pragma omp parallel
{
    b();
    #pragma omp for
    for (int i = 0; i < 10; ++i) {
        c(i);
    }
    #pragma omp critical
    {
        d();
    }
}
z();
```



You can disable waiting, so that some threads can start doing postprocessing early. This would make sense if, e.g., `d()` updates some global data structure based on what the thread computed in its own part of the parallel for loop:

```
a();
#pragma omp parallel
{
    b();
    #pragma omp for nowait
    for (int i = 0; i < 10; ++i) {
        c(i);
    }
    #pragma omp critical
    {
        d();
    }
}
z();
```



No waiting before a loop

Note that there is no synchronization point before the loop starts. If threads reach the for loop at different times, they can start their own part of the work as soon as they are there, without waiting for the other threads:

```
a();
#pragma omp parallel
{
    #pragma omp critical
    {
        b();
    }
    #pragma omp for
    for (int i = 0; i < 10; ++i) {
        c(i);
    }
    d();
}
z();
```