Programming Parallel Computers

Chapter 3: Multithreading with OpenMP

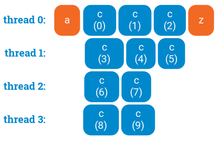
OpenMP parallel for loops

Let us start with a basic example: we would like to do some preprocessing operation `a()`, then we would like to some independent calculations `c(0)`, `c(1)`, ..., and finally some postprocessing `z()` once all calculations have finished. In this example, each of the calculations takes roughly the same amount of time. A straightforward for-loop uses only one thread and hence only one core on a multi-core computer:

```
a();
for (int i = 0; i < 10; ++i) {
    c(i);
}
z();
```

With OpenMP parallel for loops, we can easily parallelize it so that we are making a much better use of the computer. Note that OpenMP automatically waits for all threads to finish their calculations before continuing with the part that comes after the parallel for loop:

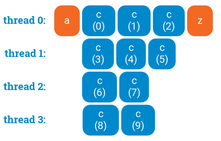
```
a();
#pragma omp parallel for
for (int i = 0; i < 10; ++i) {
    c(i);
}
z();
```



It is just a shorthand

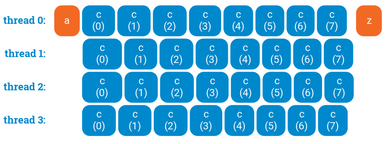
The `omp parallel for` directive is just a commonly-used shorthand for the combination of two directives: `omp parallel`, which declares that we would like to have multiple threads in this region, and `omp for`, which asks OpenMP to assign different iterations to different threads:

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



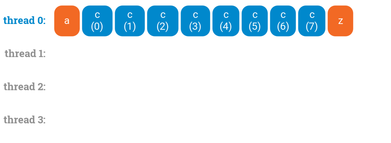
A common mistake is to just use `omp parallel` together with a for loop. This creates multiple threads for you, but it does not do any work-sharing — all threads simply run all iterations of the loop, which is most likely not what you want:

```
a();
#pragma omp parallel
for (int i = 0; i < 8; ++i) {
    c(i);
}
z();
```



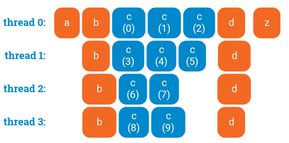
Another common mistake is to use `omp for` alone outside a parallel region. It does not do anything if we do not have multiple threads available:

```
a();
#pragma omp for
for (int i = 0; i < 8; ++i) {
    c(i);
}
z();
```



While in many cases it is convenient to combine `omp parallel` and `omp for` in one directive, please remember that you can always split them. This way it is possible to do some thread-specific preprocessing and postprocessing if needed:

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



For example, this way you can declare local variables and initialize local data structures in a convenient manner.