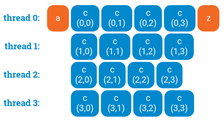
Chapter 3: Multithreading with OpenMP

Parallelizing nested loops

If we have nested for loops, it is often enough to simply parallelize the outermost loop:

```
a();
#pragma omp parallel for
for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
        c(i, j);
    }
}
z();
```



This is all that we need most of the time. You can safely stop reading this part now; in what follows we will just discuss what to do in some rare corner cases.

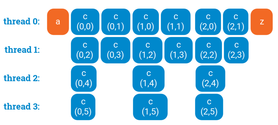
Challenges

Sometimes the outermost loop is so short that not all threads are utilized:

```
a();
#pragma omp parallel for
for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 6; ++j) {
        c(i, j);
    }
}
z();
```

We could try to parallelize the inner loop. However, then we will have more overhead in the inner loop, which is more performance-critical, and there is no guarantee that the thread utilization is any better:

```
a();
for (int i = 0; i < 3; ++i) {
    #pragma omp parallel for
    for (int j = 0; j < 6; ++j) {
        c(i, j);
    }
}
z();
```



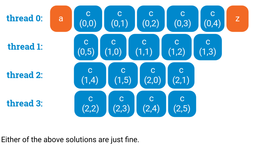
Good ways to do it

In essence, we have got here 3 × 6 = 18 units of work, and we would like to spread it evenly among the threads. The correct solution is to collapse it into one loop that does 18 iterations. We can do it manually:

```
a();
#pragma omp parallel for
for (int ij = 0; ij < 3 * 6; ++ij) {
    c(ij / 6, ij % 6);
}
z();
```

Or we can ask OpenMP to do it for us:

```
a();
#pragma omp parallel for collapse(2)
for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 6; ++j) {
        c(i, j);
    }
}
z();
```



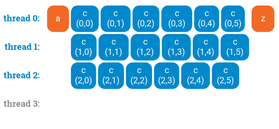
Either of the above solutions are just fine.

Wrong way to do it, part 1

Unfortunately, one often sees failed attempts of parallelizing nested for loops. This is perhaps the most common version:

```
a();
#pragma omp parallel for
for (int i = 0; i < 3; ++i) {
    #pragma omp parallel for
    for (int j = 0; j < 6; ++j) {
        c(i, j);
    }
}
z();
```

This code does not do anything meaningful. "Nested parallelism" is disabled in OpenMP by default, and the second pragma is ignored at runtime: a thread enters the inner parallel region, a team of only one thread is created, and each inner loop is processed by a team of one thread. The end result will look, in essence, identical to what we would get without the second pragma — but there is just more overhead in the inner loop:



On the other hand, if we tried to enable "nested parallelism", things would get much worse. The inner parallel region would create more threads, and overall we would have 3 × 4 = 12 threads competing for the resources of 4 CPU cores — not what we want in a performance-critical application.

Wrong way to do it, part 2

One also occasionally sees attempts of using multiple nested `omp for` directives inside one `parallel` region. This is seriously broken; OpenMP specification does not define what this would mean but simply forbids it:

```
a();
#pragma omp parallel for
for (int i = 0; i < 3; ++i) {
    #pragma omp for
    for (int j = 0; j < 6; ++j) {
        c(i, j);
    }
}
z();
```

In the system that we have been using here as an example, the above code thankfully gives a compilation error. However, if we manage to trick the compiler to compile this, e.g. by hiding the second `omp for` directives inside another function, it turns out that the program freezes when we try to run it.