Programmazione di Sistemi ~~Embedded e~~ Multicore

Teacher: Daniele De Sensi

Announcements

Announcements

- No lectures on November 19 and 20

Recap

Recap

- OpenMP basics
- OpenMP scope

Parallel For

Parallel for

- Forks a team of threads to execute the following structured block.
- However, the structured block following the parallel for directive must be a for loop.
- Furthermore, with the parallel for directive the system parallelizes the for loop by dividing the iterations of the loop among the threads.

Trapezoid Example

```
h = (b-a)/n;
approx = (f(a) + f(b))/2.0;
for (i = 1; i <= n-1; i++)
   approx += f(a + i*h);
approx = h*approx;
```

```
    h = (b-a)/n;
    approx = (f(a) + f(b))/2.0;
#   pragma omp parallel for num_threads(thread_count) \
       reduction(+: approx)
    for (i = 1; i <= n-1; i++)
       approx += f(a + i*h);
    approx = h*approx;
```

Legal forms for parallelizable for statements

```
                                            index++
                                            ++index
                      index < end           index--
                      index <= end          --index
for ( index = start ; index >= end ;        index += incr            )
                      index > end           index -= incr
                                            index = index + incr
                                            index = incr + index
                                            index = index - incr
```

Why? It allows the runtime system to determine the number of iterations prior to the execution of the loop

Caveats

- The variable index must have integer or pointer type (e.g., it can't be a float).
- The expressions start, end, and incr must have a compatible type. For example, if index is a pointer, then incr must have integer type.
- The expressions start, end, and incr must not change during execution of the loop.
- During execution of the loop, the variable index can only be modified by the "increment expression" in the for statement.

examples

```
for (i=0; i<n; i++) {
   if (…) break;     //cannot be parallelized
}
```

```
for (i=0; i<n; i++) {
   if (…) return 1; //cannot be parallelized
}
```

```
for (i=0; i<n; i++) {
   if (…) exit();     //can be parallelized
}
```

```
for (i=0; i<n; i++) {
   if (…) i++;     //CANNOT be parallelized
}
```

Questions?

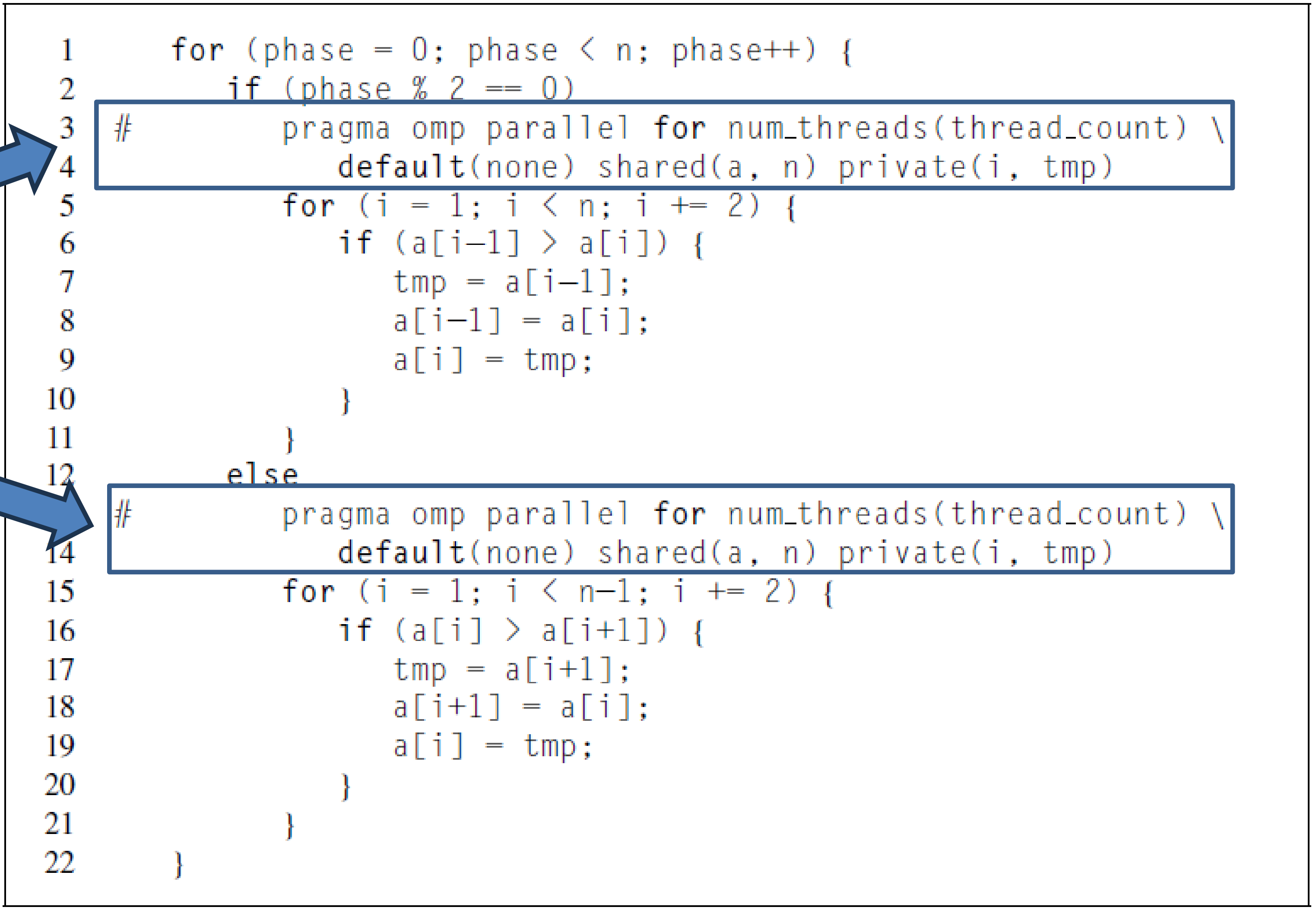
Example: Odd-Even Sort

Odd-Even Sort

This might fork/join new threads everytime it is called (depends on the implementation)

If it does so, we would have some **overhead**

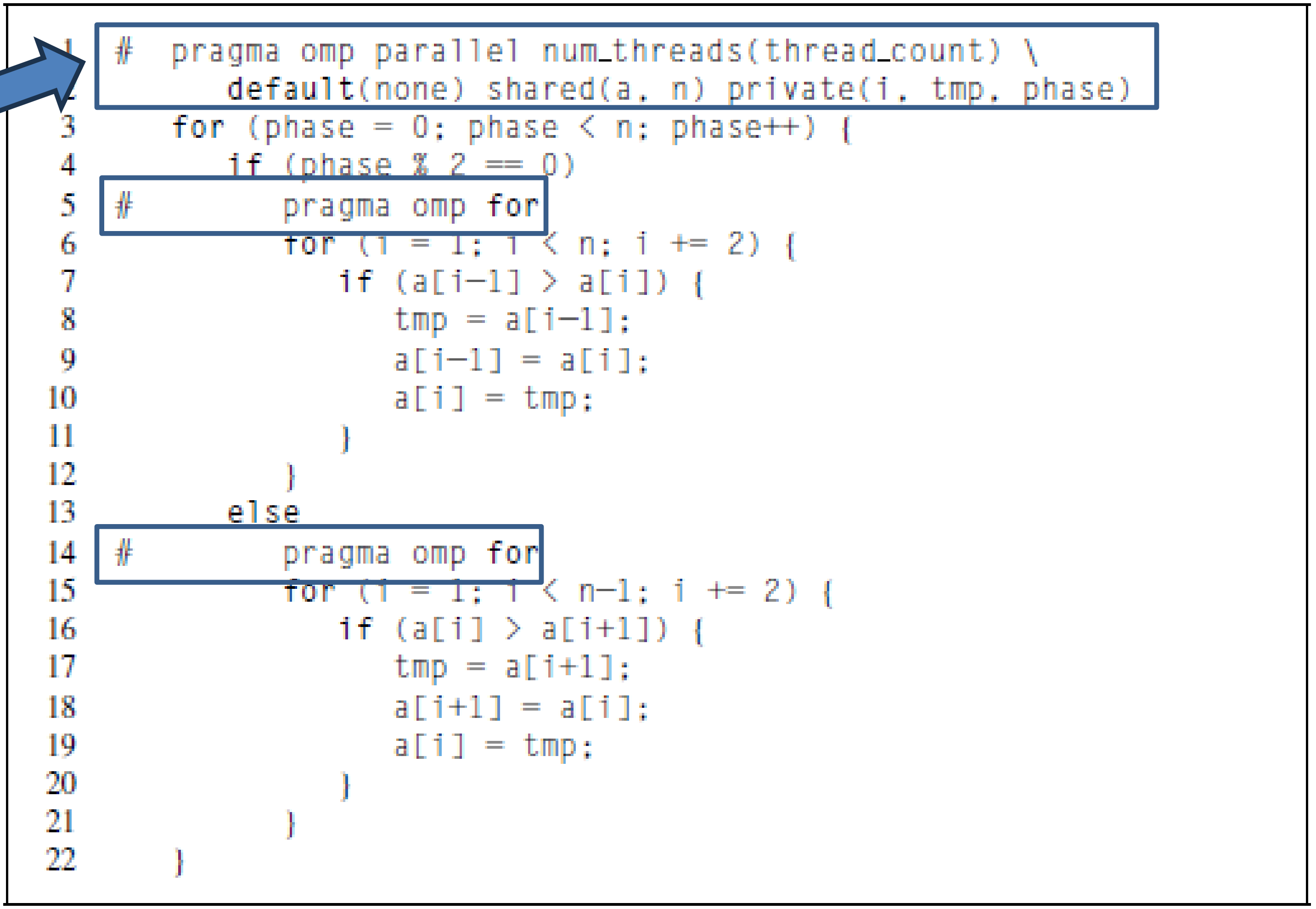
Can we just create the threads at the beginning (before line 1)?



```
1     for (phase = 0; phase < n; phase++) {
2        if (phase % 2 == 0)
3  #        pragma omp parallel for num_threads(thread_count) \
4              default(none) shared(a, n) private(i, tmp)
5           for (i = 1; i < n; i += 2) {
6              if (a[i−1] > a[i]) {
7                 tmp = a[i−1];
8                 a[i−1] = a[i];
9                 a[i] = tmp;
10             }
11          }
12       else
13 #        pragma omp parallel for num_threads(thread_count) \
14             default(none) shared(a, n) private(i, tmp)
15          for (i = 1; i < n−1; i += 2) {
16             if (a[i] > a[i+1]) {
17                tmp = a[i+1];
18                a[i+1] = a[i];
19                a[i] = tmp;
20             }
21          }
22    }
```

Odd-Even Sort

Fork threads
only here



```
#  pragma omp parallel num_threads(thread_count) \
      default(none) shared(a, n) private(i, tmp, phase)
   for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
#        pragma omp for
         for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
               tmp = a[i-1];
               a[i-1] = a[i];
               a[i] = tmp;
            }
         }
      else
#        pragma omp for
         for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
               tmp = a[i+1];
               a[i+1] = a[i];
               a[i] = tmp;
            }
         }
   }
```

Odd-Even Sort

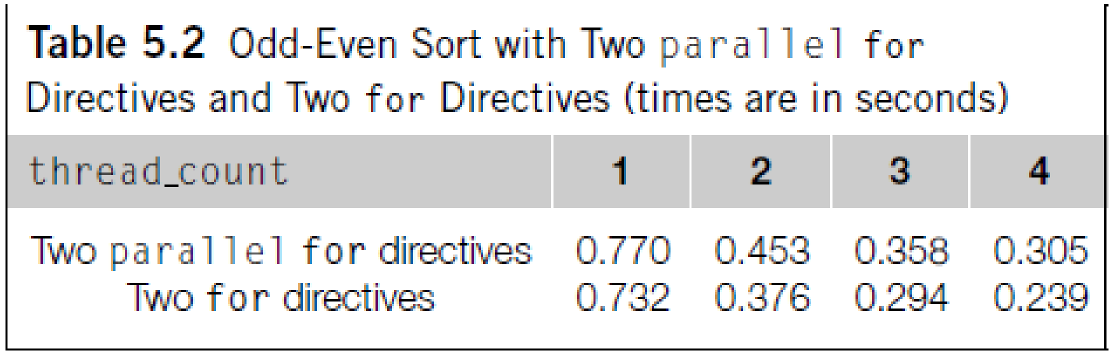


Table 5.2 Odd-Even Sort with Two `parallel for` Directives and Two `for` Directives (times are in seconds)

`thread_count`	**1**	**2**	**3**	**4**
Two `parallel for` directives	0.770	0.453	0.358	0.305
Two `for` directives	0.732	0.376	0.294	0.239

Reusing the same threads provide faster execution times

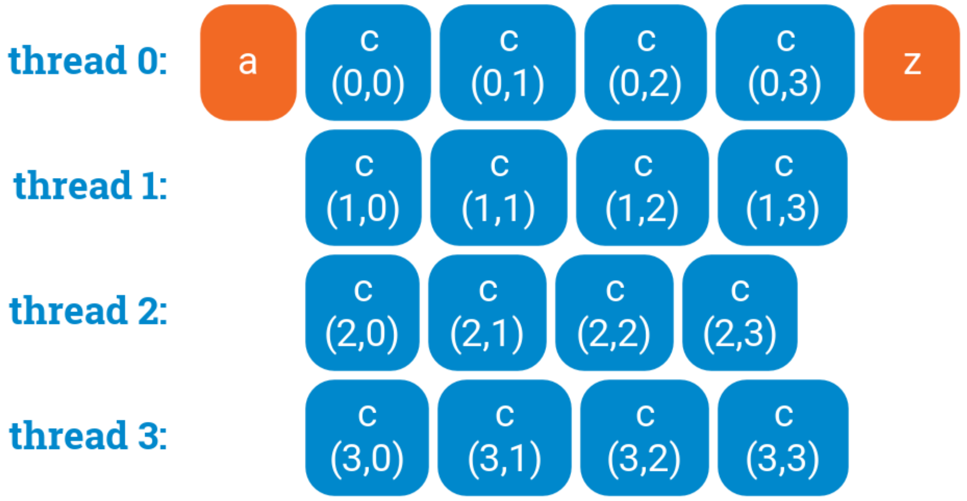
Questions?

Nested Loops

Nested for 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();
```



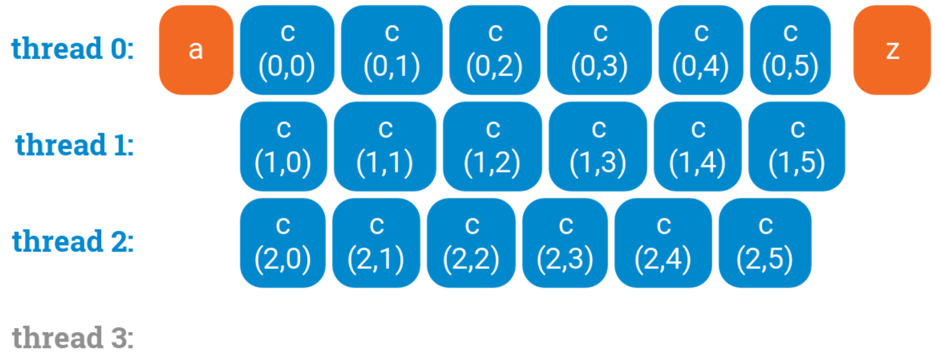
https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

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

3 iterations, so it won't have sense to start more than 3 threads

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

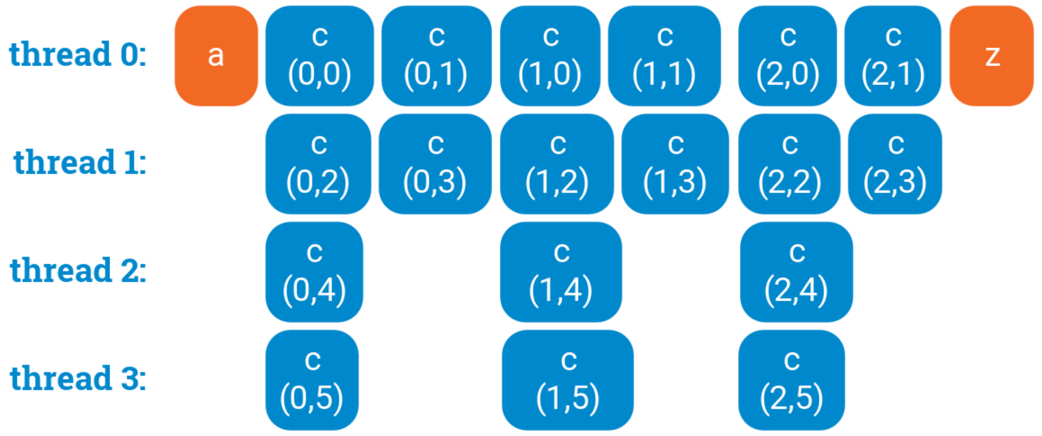


https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

- We could try to parallelize the inner loop, but there is no guarantee that the thread utilization is better

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



https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

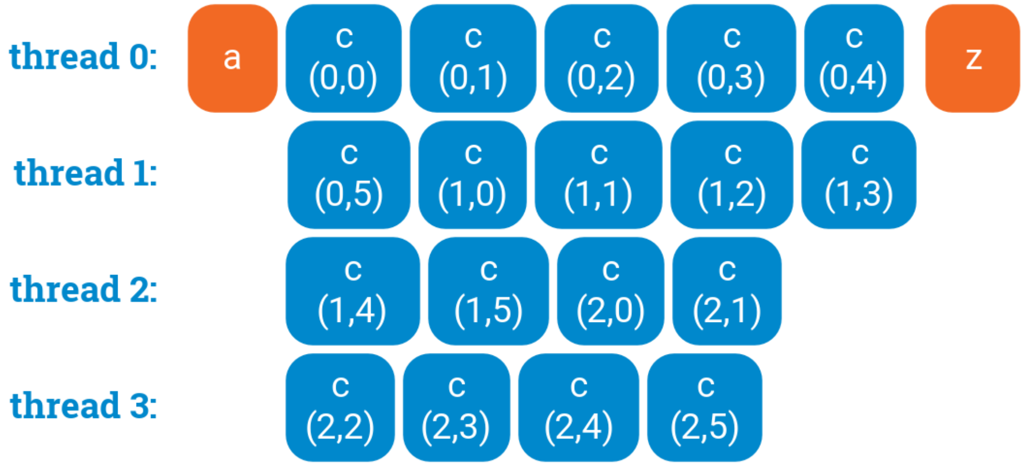
- The correct solution is to **collapse it into one loop** that does 18 iterations. We can do it manually:

https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

- 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();
```

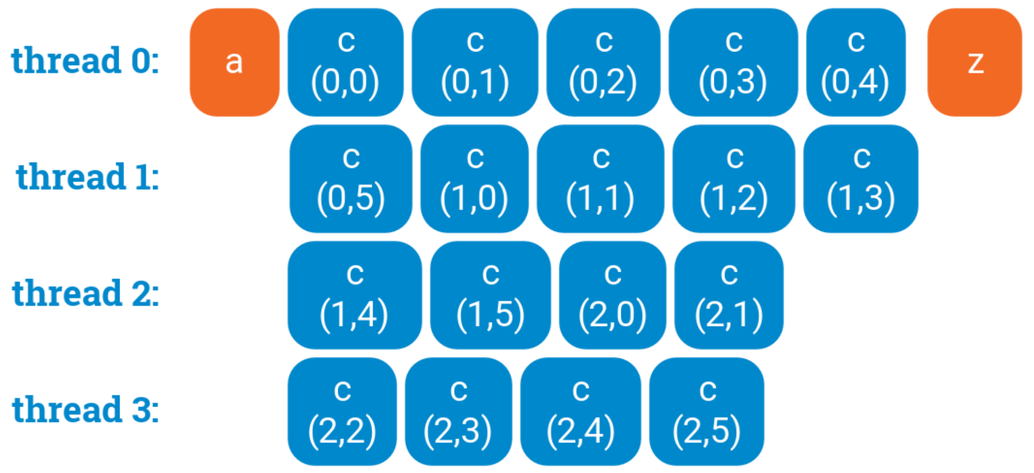


https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

- 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();
```

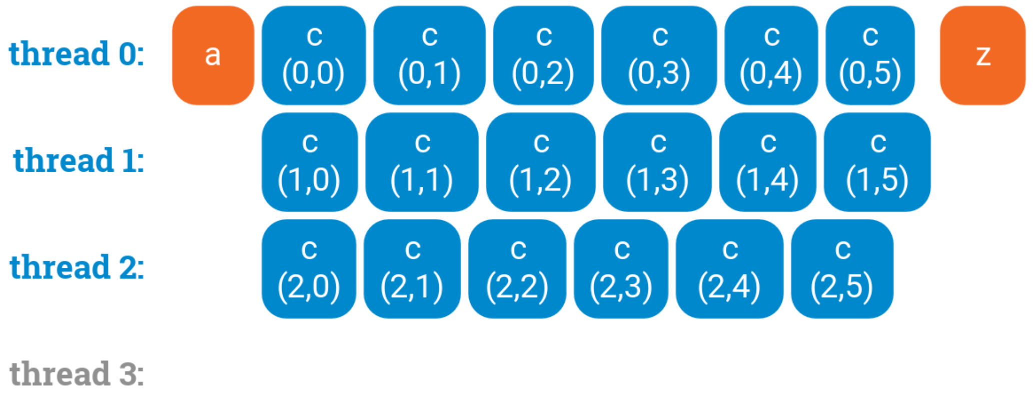


https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

- Wrong way: "Nested parallelism" is disabled in OpenMP by default (i.e., inner parallel for pragmas will be ignored)

```
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();
```



https://ppc.cs.aalto.fi/ch3/nested/

Nested for loops

- Wrong way: If "Nested parallelism" is enabled it will create 12 threads on a server with 4 cores (3*4)!

```
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();
```

https://ppc.cs.aalto.fi/ch3/nested/

OPIS

OPIS

B76Z7JSO

Exercises

How to take timings

- Use the GET_TIME macro we have seen last week (you can find the code on Github)
- Use omp_get_wtime