SaC lab code generation

Task 1: Code Specialisation

Describe the potential and the difficulties of code specialisation. Give examples / describe application scenarios for all pros and challenges you explain.

Task 2: Code Optimisation

Name and explain 3 different code optimisations that are implemented in the SaC tool chain. Try to present an example code that benefits from all these 3 optimisations and show the resulting optimised code.

Task 3: Concurrency Pattern

Consider the following implementation of addition in SaC:

```
double add (double a, double b)
{
    return a + b;
}

double[m,n:shp] add (double[m,n:shp] a, double[m,n:shp] b)
{
    return { [i] -> add (a[i], b[i]) };
}
```

Assuming no further code optimisations, what kind of concurrency pattern would a call to `add` with two arrays of shape `[3,2]` lead to? Draw a fork-join diagram in the style used in the lecture to depict the pattern. Bear in mind that non-scalar selection is defined in terms of `with-loops` as well!

Task 4

Reconsider the previous task. Let us now assume that we are dealing with more realistic problem sizes, e.g. shapes of `[3000,2000]` instead of `[3,2]`; what potential challenges in terms of parallel performance would you anticipate? Try to identify at least one optimisation that could be applied to tackle the anticipated challenge and describe its impact on the code. How does the optimised code look like and how does the resulting concurrency pattern look like? Present the new pattern using the original shape of `[3,2]` once more.

Hand-out: 6/03/2025 Hand-in: 13/03/2025