DaCe Workshop III/23

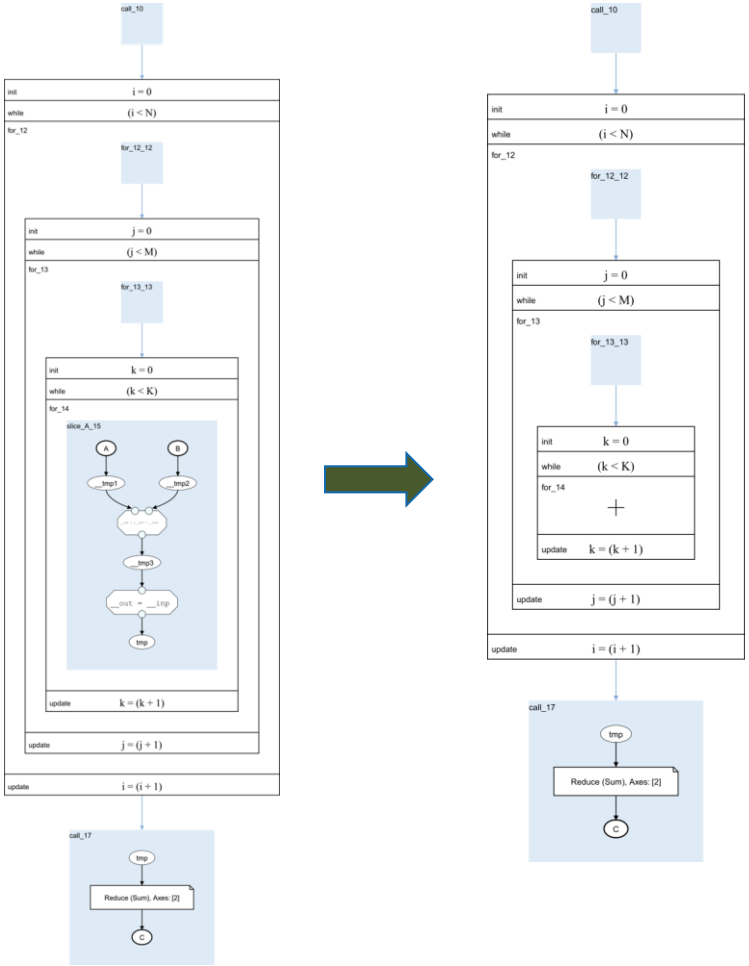
20.10.2023

Loops as First Class Citizens

Loops as First Class Citizens

We will try to do this by annotating loop information on loop guards. Be failsafe, i.e., if something changes by a transformation, discard this information if it's involved. If we ever have to run loop detection, re-annotate this. But initially, the frontend should annotate this information.

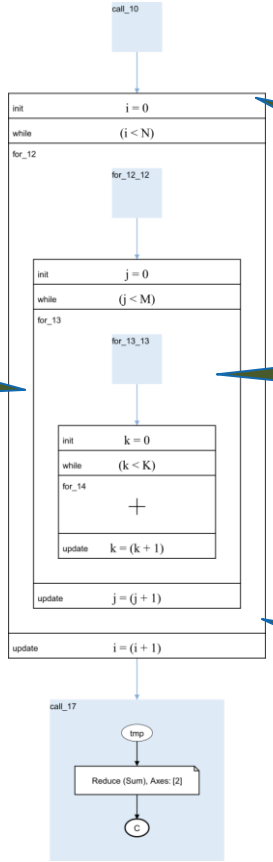
Good Progress



This is Exciting

Better visualization and comprehension

Easier interaction and analysis



Show data dependencies for groups of states

Show inter-iteration dependencies for loops

Invariants and Pre-/Post-Conditions, drag-and-drop reordering, 1-click parallelization, ….

Initial Approach Insufficient

- **Keep state machine unchanged**
- **Annotate loop information on:**
  - Loop guard
  - SDFG

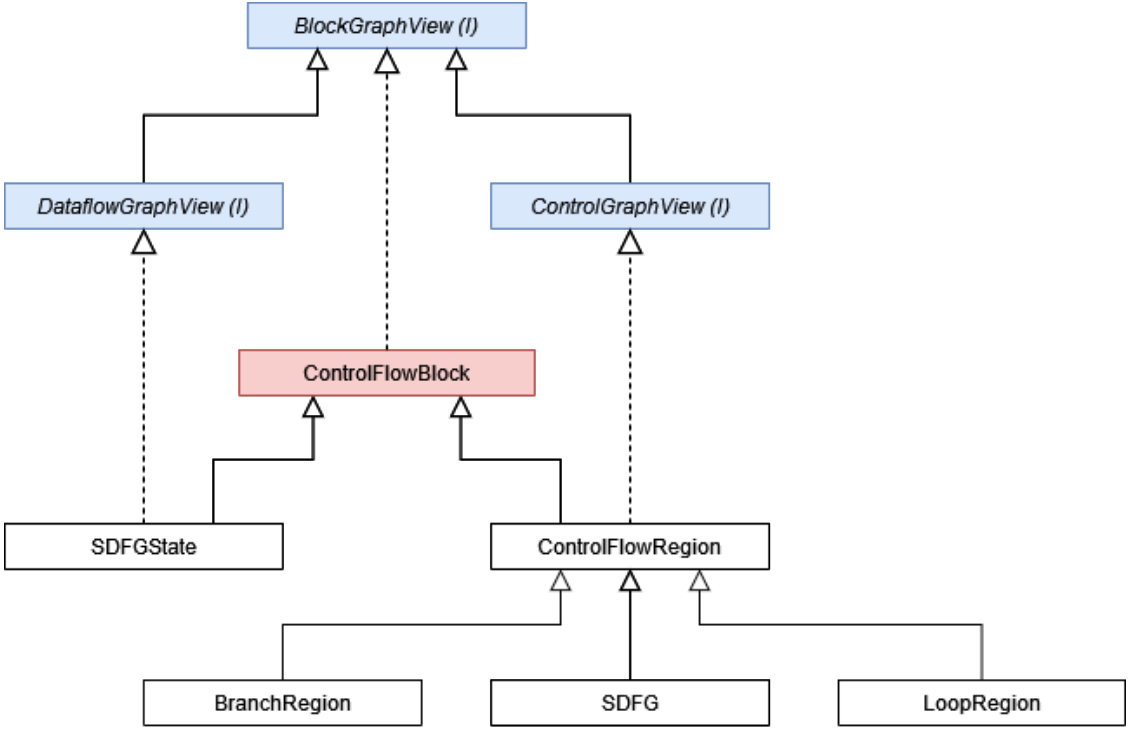
\+ **Minimally invasive, no breaking API changes**

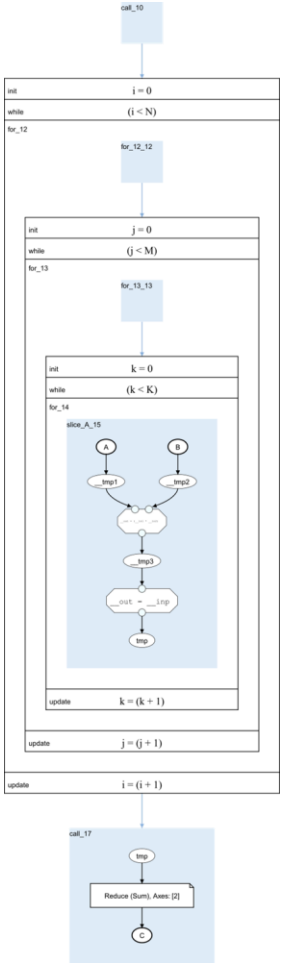
– **Insufficiently robust, burden of maintenance**

More Invasive Solution

- Run SDFG into hierarchical control flow graph
- Loops are separate kinds of state machine nodes
- Changes the SDFG structure

New SDFG Structure





Nested Datatypes

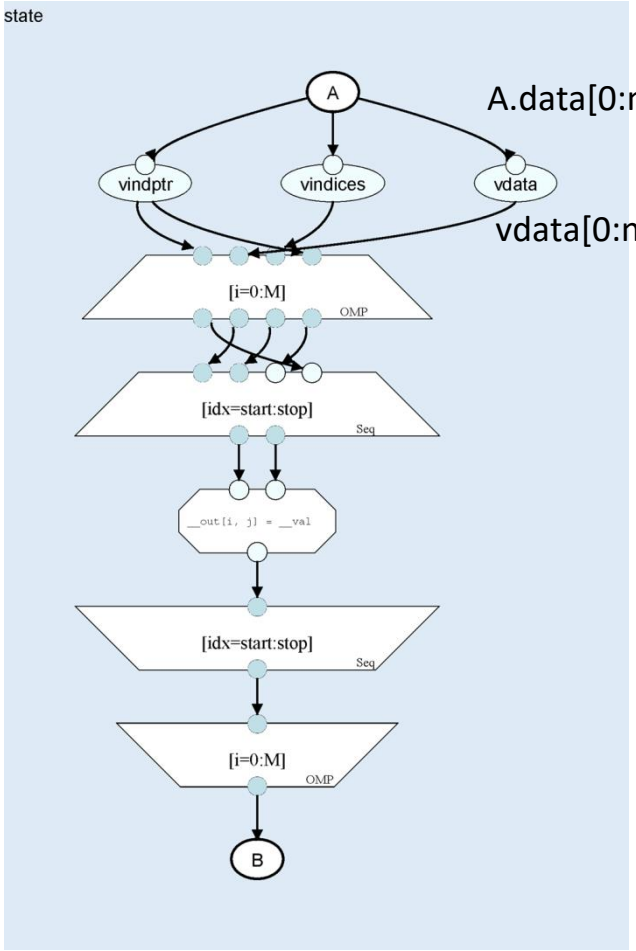
Idea: Data can contain data

The benefit of this would be the ability to express sparse data structures like CSRArrays as a single data container that contains other data containers (e.g. rows array, cols array, data array, etc.). The corresponding access nodes have a connector that indicates which sub data is being accessed.

This can be done multiple times (nested).

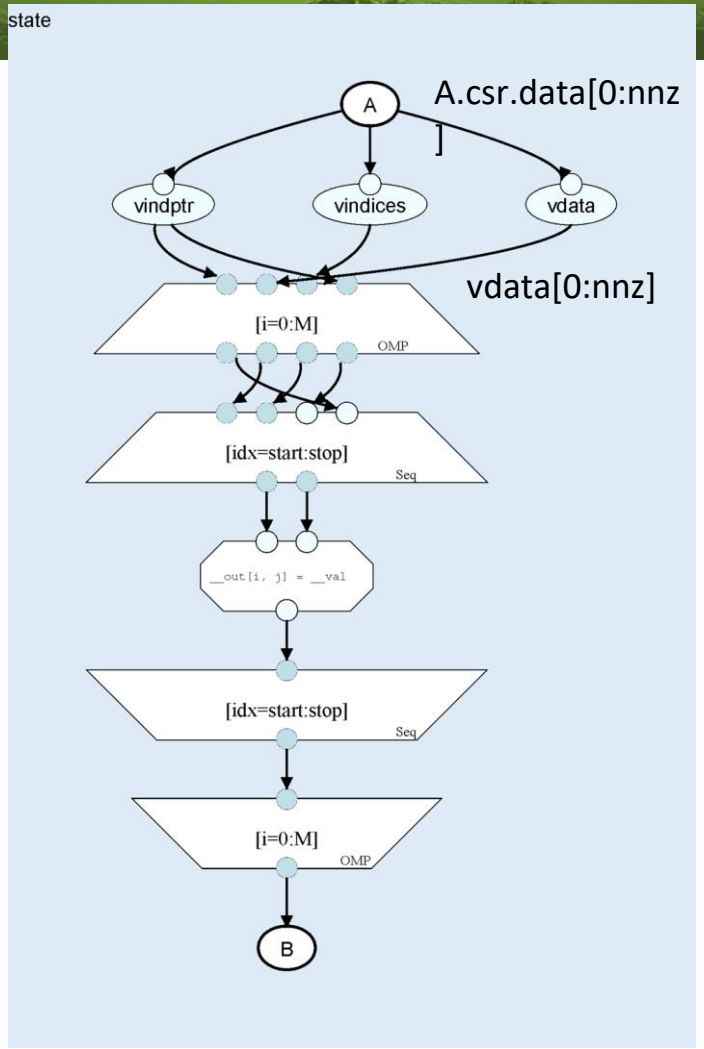
Alternative name should be a struct, the nested data types name is confusing.

1st Approach



Nested data are first "dereferenced" to views.

1st Approach



Multi-un-nesting occurs at the edge from the Structure to the View.

Current Developments: Basic Frontend Support

```
M, N, nnz = (dace.symbol(s) for s in ('M', 'N', 'nnz'))
CSR = dace.data.Structure(
    dict(indptr=dace.int32[M + 1],
         indices=dace.int32[nnz],
         data=dace.float32[nnz]),
    name='CSRMatrix')

@dace.program
def csr_to_dense_python(A: CSR, B: dace.float32[M, N]):
    for i in dace.map[0:M]:
        for idx in dace.map[A.indptr[i]:A.indptr[i + 1]]:
            B[i, A.indices[idx]] = A.data[idx]
```

Offset normalization in Fortran

- **First pass: parent scope assigner.**
  - Not available previously.
  - Assigns to each AST node its current scope.

- **Second pass: scope variable mapper.**
  - Not available previously.
  - In Fortran AST, references to variables in *execution part* are not automatically linked to type information stored in *specification part*.
  - Create a mapping (scope, variable) -> type information.

- **Third pass: offset normalizer.**
  - For each array access, look up the corresponding type information and adjust the address.

- **Result: all arrays have 0-based indexing from DaCe PoV.**