Alexandru Calotoiu

DaFlEx

C2DaCe challenges

- Classes
  - Inheritance
  - Contexts
- Recursions
  - Tail recursion
  - Indirect recursion
- Pointers
  - Unrestrictred arithmetic
- Stateful library calls
  - Automatic assessment
- Template programming
- Library nodes
- Encapsulation

F2DaCe challenges

- Generalized views
- Vector operations
- Labels & GoTo’s
- Intrinsic function coverage
- Modern Fortran

DaCe challenges

- Application-level ToGPU transform
  - + Associated transforms

Engineering efforts

Research efforts

Collaboration?

Application-level ToGPU transform

- **Schedule tree representation**
  - New IR
  - Allows for different transformation to happen efficiently
  - Work in Progress

```
map i in [0:N]: __tmp2[0:5, 0:5] = library MatMul[alpha=1, beta=0](A[i, 0:5, 0:5], B[i, 0:5, 0:5]) map __i0, __i1 in [0:5, 0:5]:
__return[i, __i0, __i1] = tasklet(cst[0], __tmp2[__i0, __i1])
```

Other topics

- **Merging Fortran frontend**
  - (almost finished)
- **Snitch backend**
  - Development in progress
  - Basic C works
  - Snitch extensions not finished
- **Read the docs:**
  - https://spcldace.readthedocs.io/en/latest/

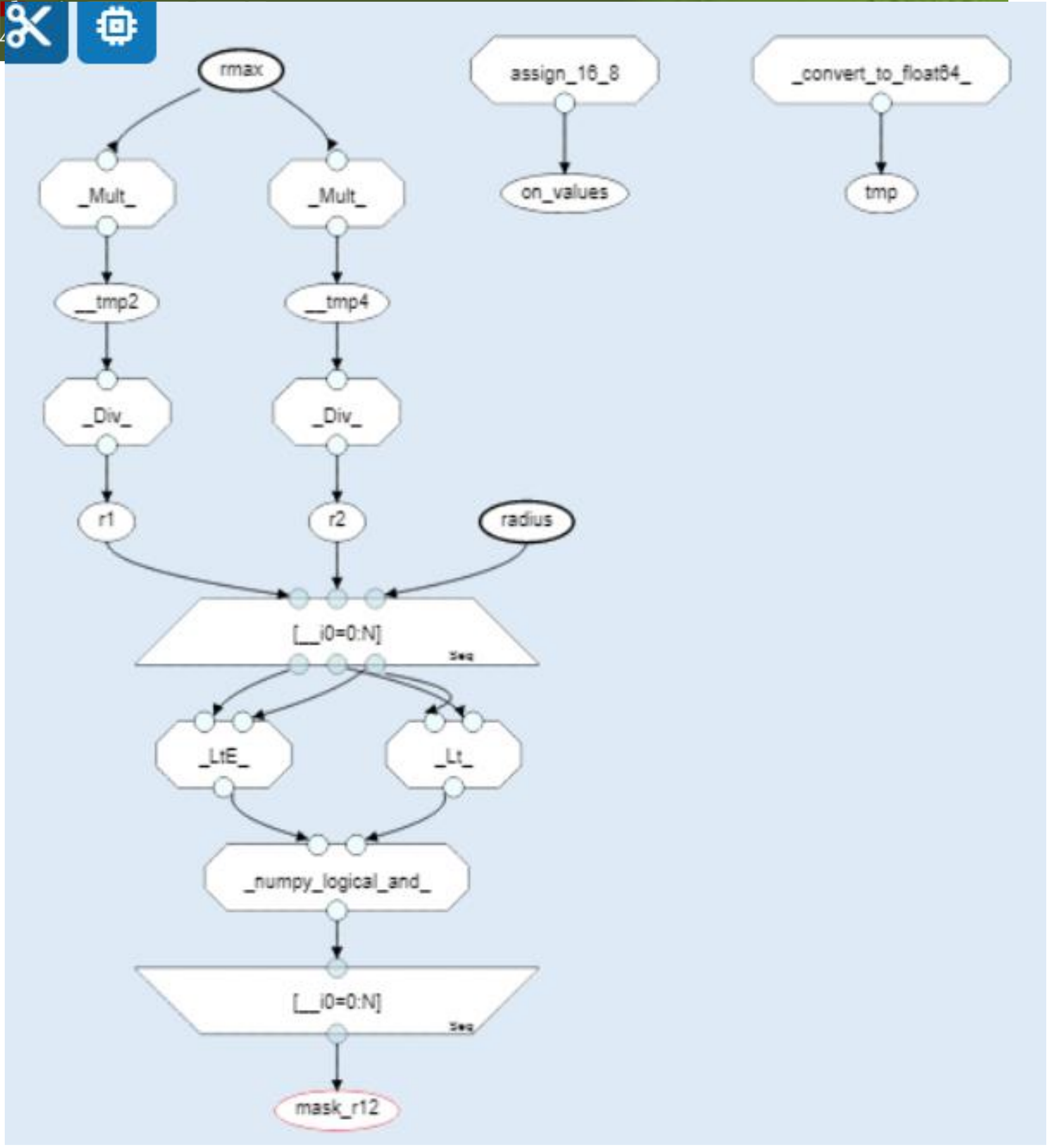
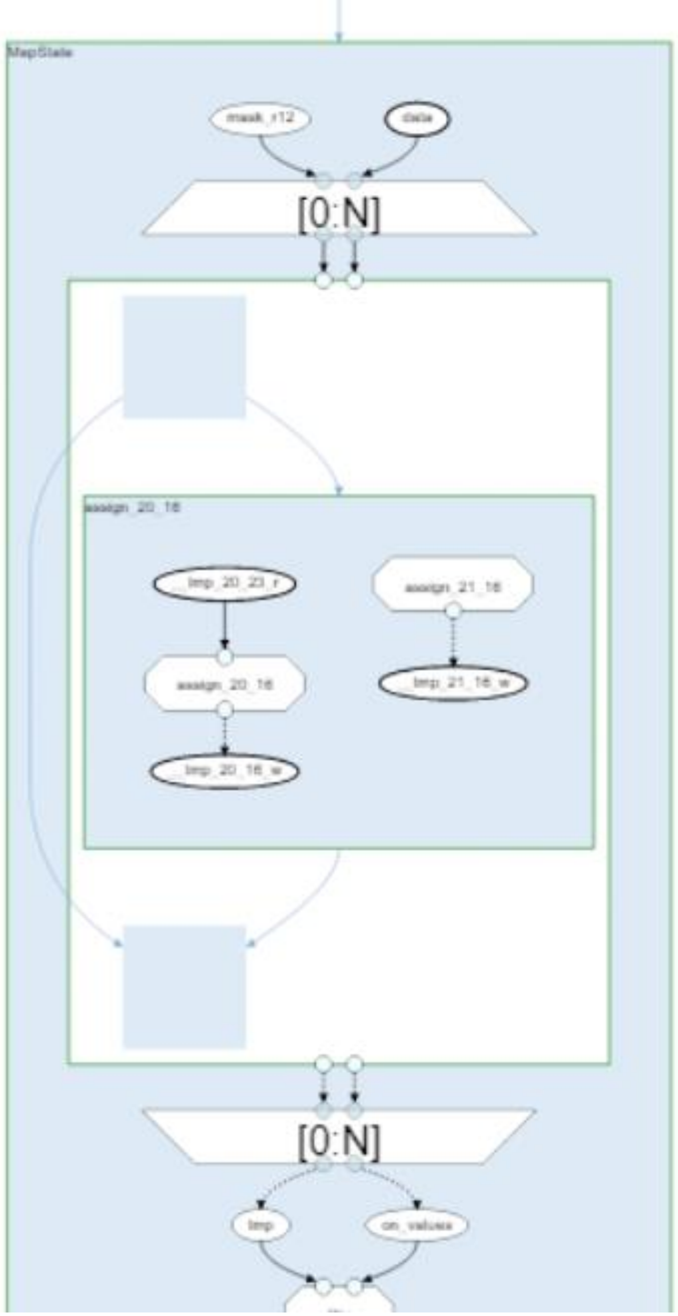
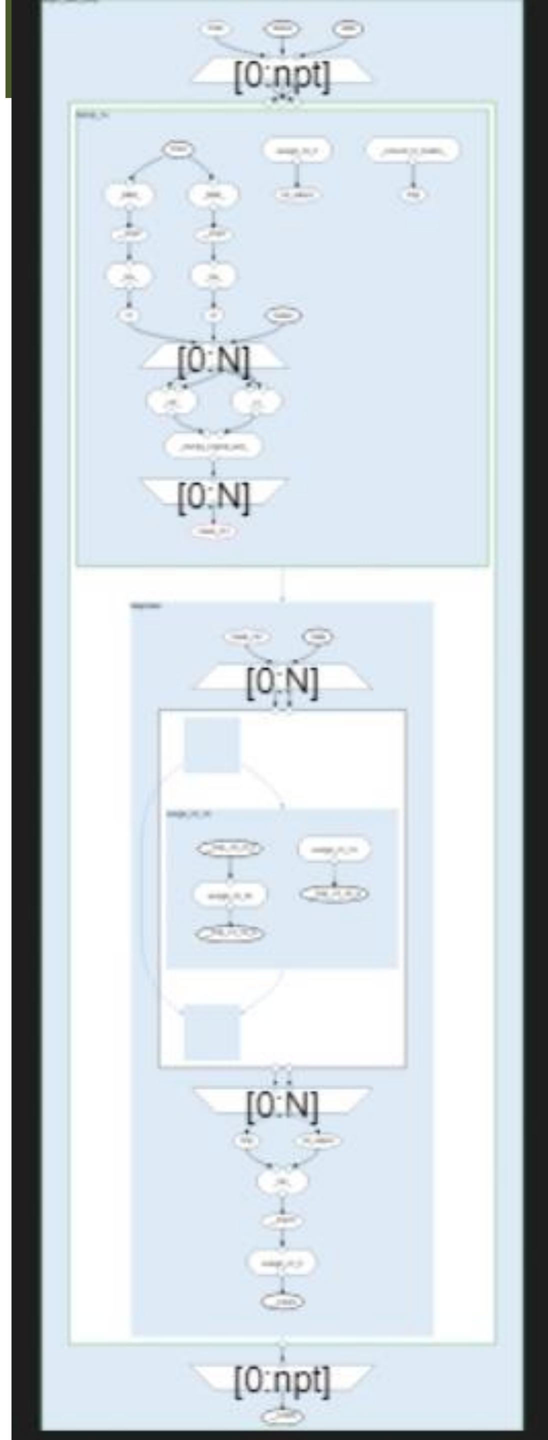
IR Lemmas

If there exists a bi-directional transformation with no loss of information between two representations, then any transformation that is possible in one, must be possible in the other

Corollary: if the engineering or runtime complexity of a transformation in one representation is larger than translating back and forth and applying the transformation in another representation, it should be done there.

Code example

```python
@dace.program
def dace_azimint_naive(data: dace.float64[N], radius:
dace.float64[N]):
    rmax = np.amax(radius)
    res = np.zeros((npt, ), dtype=np.float64)
    for i in range(npt):
        r1 = rmax * i / npt
        r2 = rmax * (i + 1) / npt
        mask_r12 = np.logical_and((r1 <= radius), (radius < r2))
        on_values = 0
        tmp = np.float64(0)
        for j in dace.map[0:N]:
            if mask_r12[j]:
                tmp += data[j]
                on_values += 1
        res[i] = tmp / on_values
    return res
```


rmax
assign_16_8
_convert_to_float64_
Mult
Mult
on_values
tmp
__tmp2
__tmp4
Div
Div
r1
r2
radius
[__i0=0:N]
Seq
LtE
_Lt_
_numpy_logical_and_
[__i0=0:N]
Seq
mask_r12

MapState
mask_r12
data
[0:N]
assign_20_16
__tmp_20_23_r
assign_21_16
assign_20_16
__tmp_21_16_w
__tmp_20_16_w
[0:N]
tmp
on_values

[0:npt]
[0:N]
[0:N]
[0:N]
[0:N]
[0:npt]

```
@dace.program
def dace_azimint_naive(data: dace.float64[N], radius:
dace.float64[N]):
    rmax = np.amax(radius)
    res = np.zeros((npt, ), dtype=np.float64)
    for i in range(npt):
        1 r1 = rmax * i / npt
        2 r2 = rmax * (i + 1) / npt
        3 mask_r12 = np.logical_and((r1 <= radius), (radius <
r2))
        4 on_values = 0
        5 tmp = np.float64(0)
        6 for j in dace.map[0:N]:
            if mask_r12[j]:
                tmp += data[j]
                on_values += 1
        res[i] = tmp / on_values
    return res
```

It is good for maps that have (a) the same range and (b) can pipe the output of one to the input of another to be together.

State Fusion and WCR Edge Representation

- Change Map representation
- Add input wcr-style edge
- Increase “fusibility”