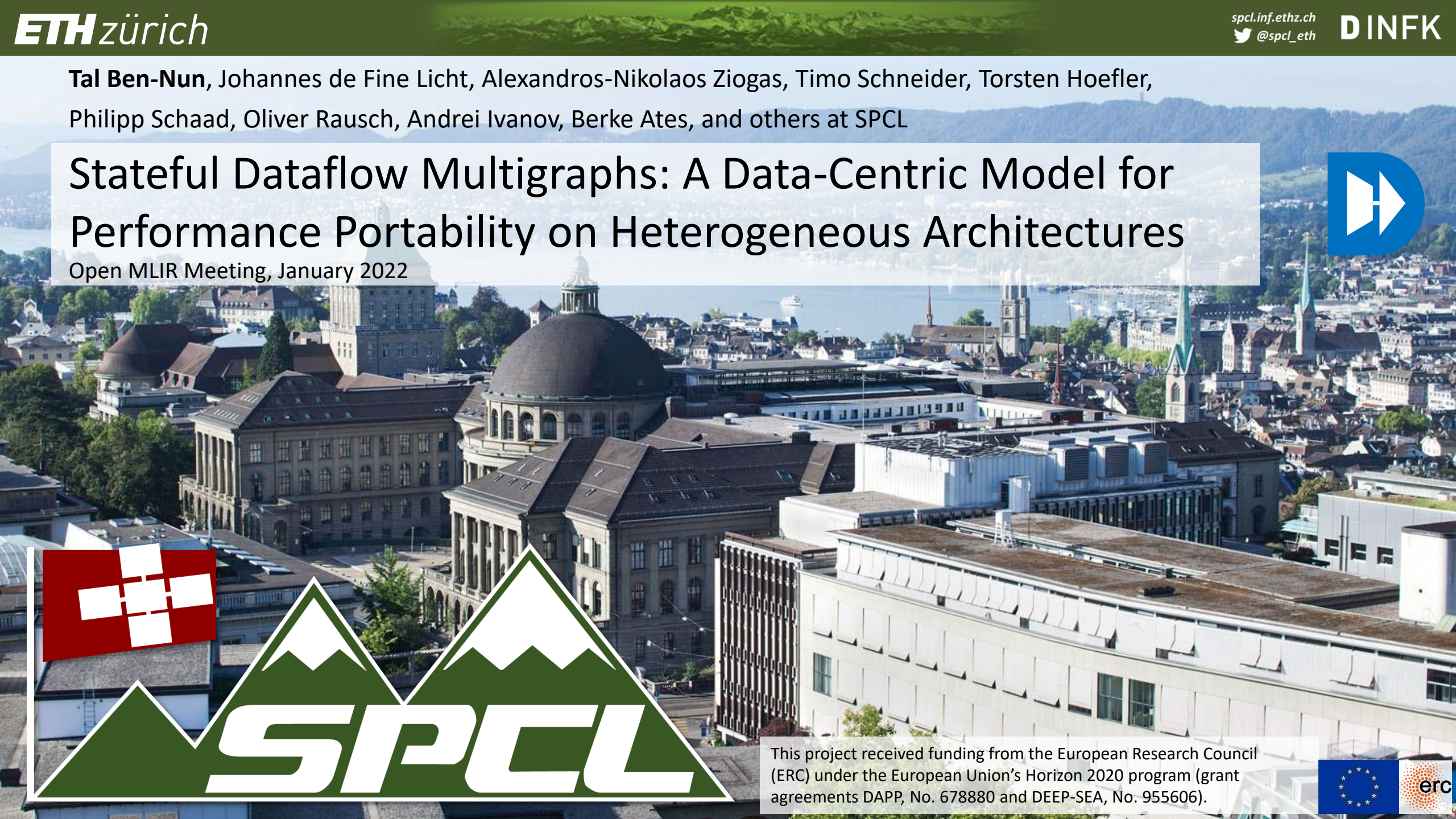
Tal Ben-Nun, Johannes de Fine Licht, Alexandros-Nikolaos Ziogas, Timo Schneider, Torsten Hoefler,
Philipp Schaad, Oliver Rausch, Andrei Ivanov, Berke Ates, and others at SPCL

Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures

Open MLIR Meeting, January 2022



This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 program (grant agreements DAPP, No. 678880 and DEEP-SEA, No. 955606).

Domain Scientist

```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

(or `C += A @ B`)

333 LoC

System

```
__syncthreads();

// Compute a grid of C matrix tiles in each warp.
#pragma unroll
for (int k_step = 0; k_step < CHUNK_K; k_step++) {
    wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a[WARP_COL_TILES];
    wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b[WARP_ROW_TILES];

    #pragma unroll
    for (int i = 0; i < WARP_COL_TILES; i++) {
        size_t shmem_idx_a = (warpId/2) * M * 2 + (i * M);
        const half *tile_ptr = &shmem[shmem_idx_a][k_step * K];

        wmma::load_matrix_sync(a[i], tile_ptr, K * CHUNK_K + SKEW_HALF);

        #pragma unroll
        for (int j = 0; j < WARP_ROW_TILES; j++) {
            if (i == 0) {
                // Load the B matrix fragment once, because it is going to be reused
                // against the other A matrix fragments.
                size_t shmem_idx_b = shmem_idx_b_off + (WARP_ROW_TILES * N) * (warpId%2)
                                   + (j * N);
                const half *tile_ptr = &shmem[shmem_idx_b][k_step * K];

                wmma::load_matrix_sync(b[j], tile_ptr, K * CHUNK_K + SKEW_HALF);
            }

            wmma::mma_sync(c[i][j], a[i], b[j], c[i][j]);
        }
    }
}

__syncthreads();
```

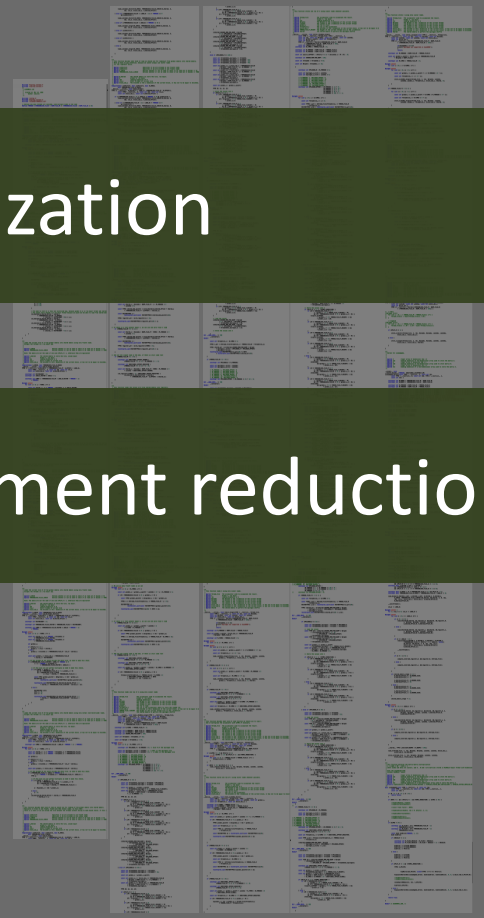
Tensor Core NVIDIA Code Sample

Domain Scientist

```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

(or `C += A @ B`)

System



1,717 lines

COSMA on CUDA/HIP

Same code for algorithm and optimization

High-performance optimization = data movement reduction

Kwasniewski et al., “Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication” SC’19

DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

Python | DSLs

PyTorch | MATLAB

...

Data-Centric Program

Performance Engineer

Data-Centric Intermediate Representation (SDFG)

Graph Transformations

Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

CPU Binary

GPU Binary

FPGA Modules

Data-Centric Guiding Principles

1. Separate data containers from computation

2. Data movement as a first-class citizen
   (dependencies → program order)

```
B = A * 5
D = C + 1
```

3. Control dependencies *only when dataflow is not implied*

```
B = A * 5
A = C + 1
```

4. Coarsening: multi-level view of data movement

DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

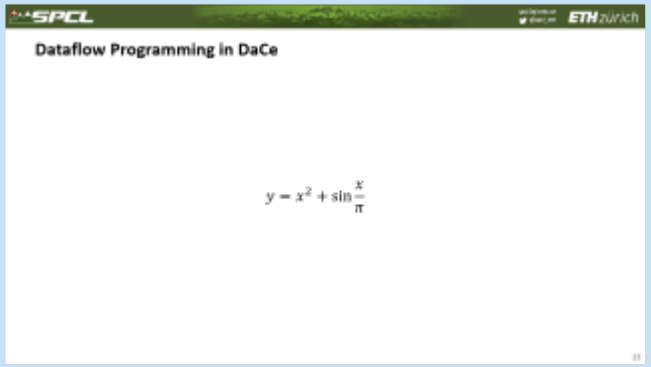
Python | DSLs

PyTorch | MATLAB

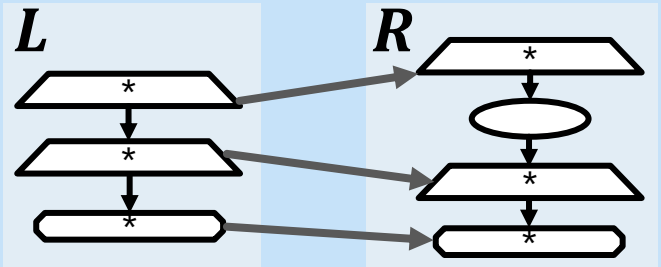
...

Data-Centric Program

Performance Engineer



Data-Centric Intermediate Representation (SDFG)



Graph Transformations

Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

CPU Binary

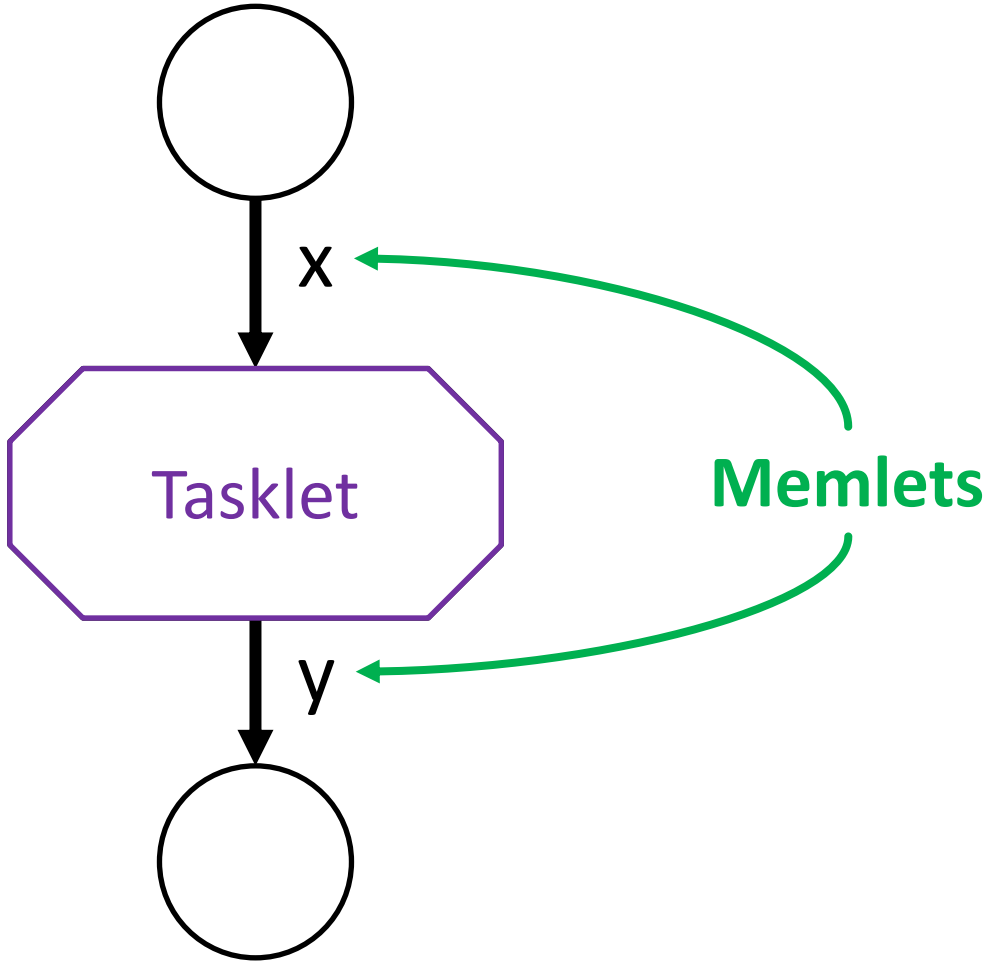
GPU Binary

FPGA Modules

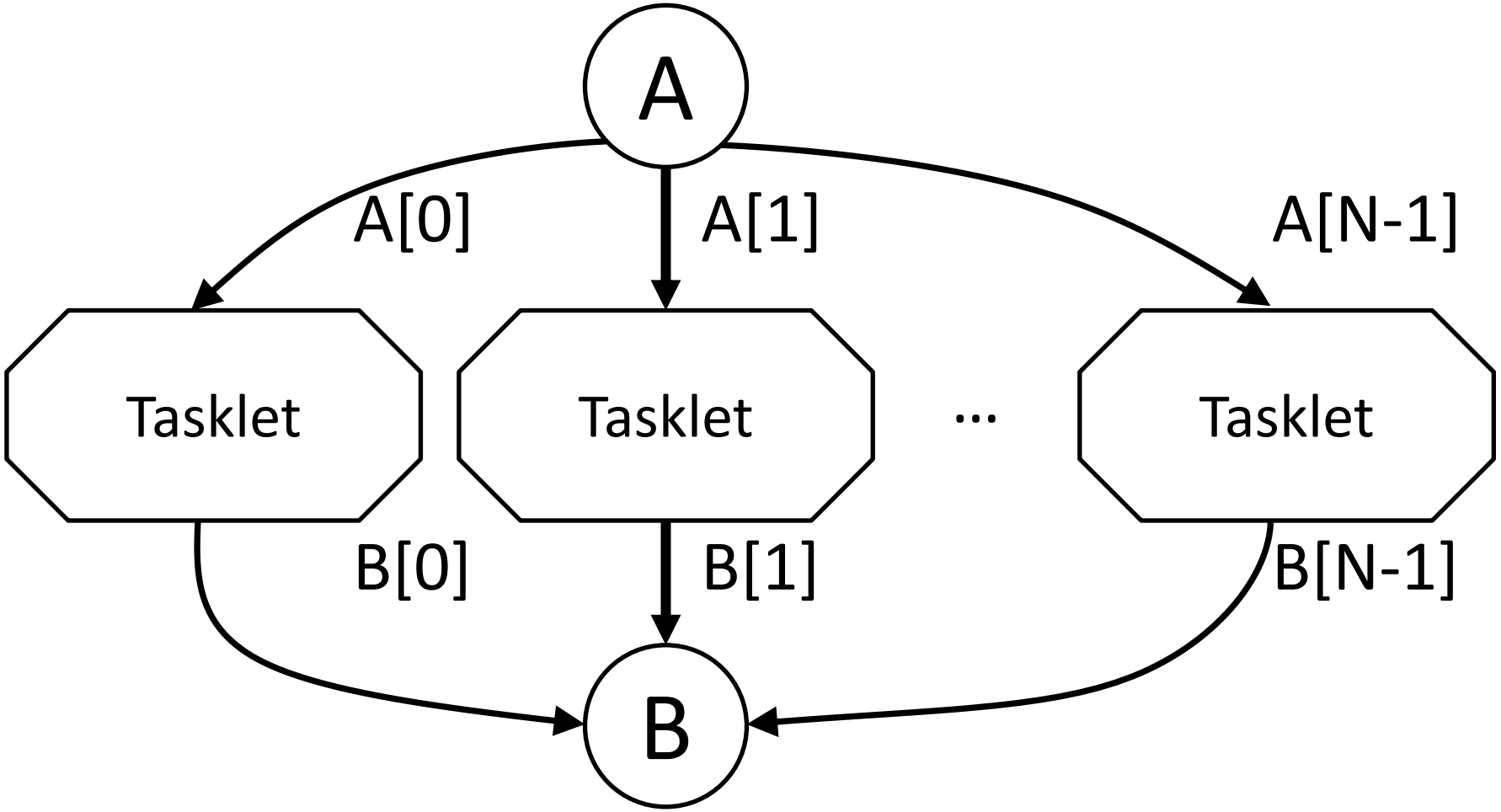
Dataflow Programming in DaCe

$$y = x^2 + \sin\frac{x}{\pi}$$

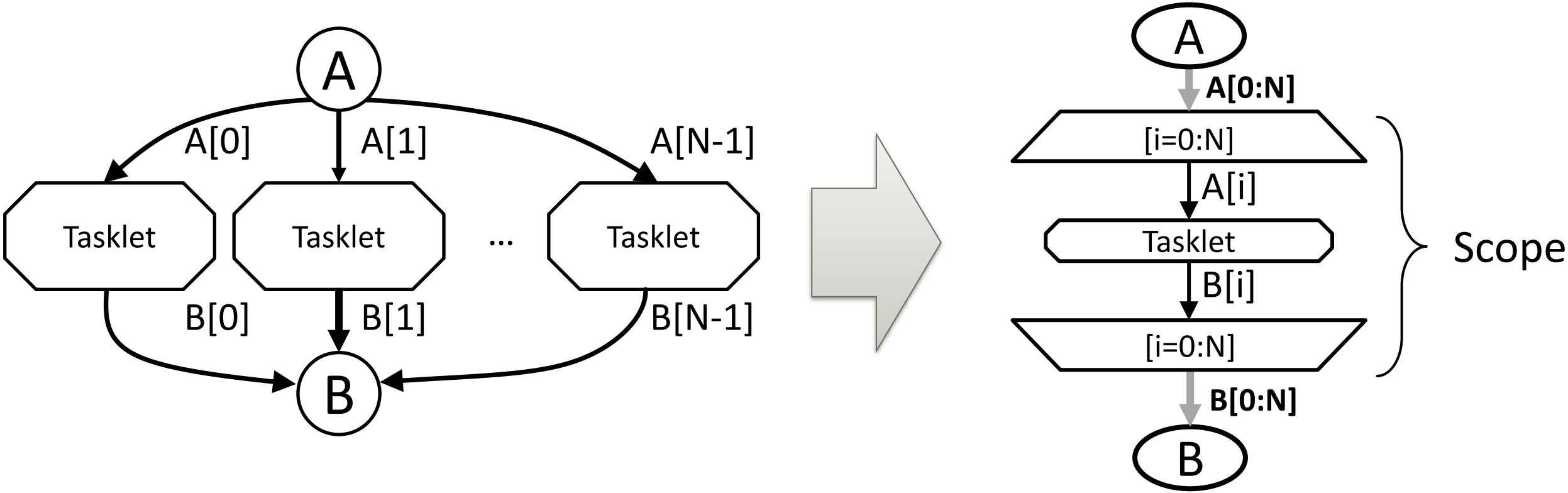
Dataflow Programming in DaCe



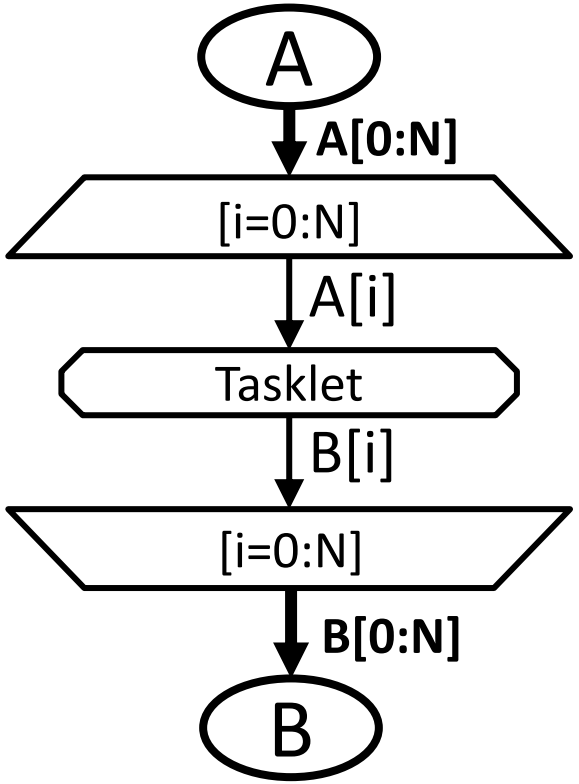
Parallel Dataflow Programming

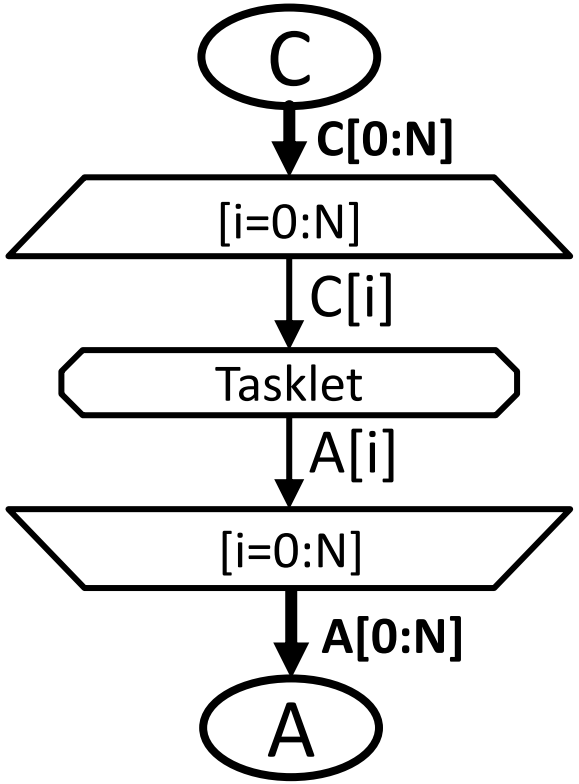


Parallel Dataflow Programming

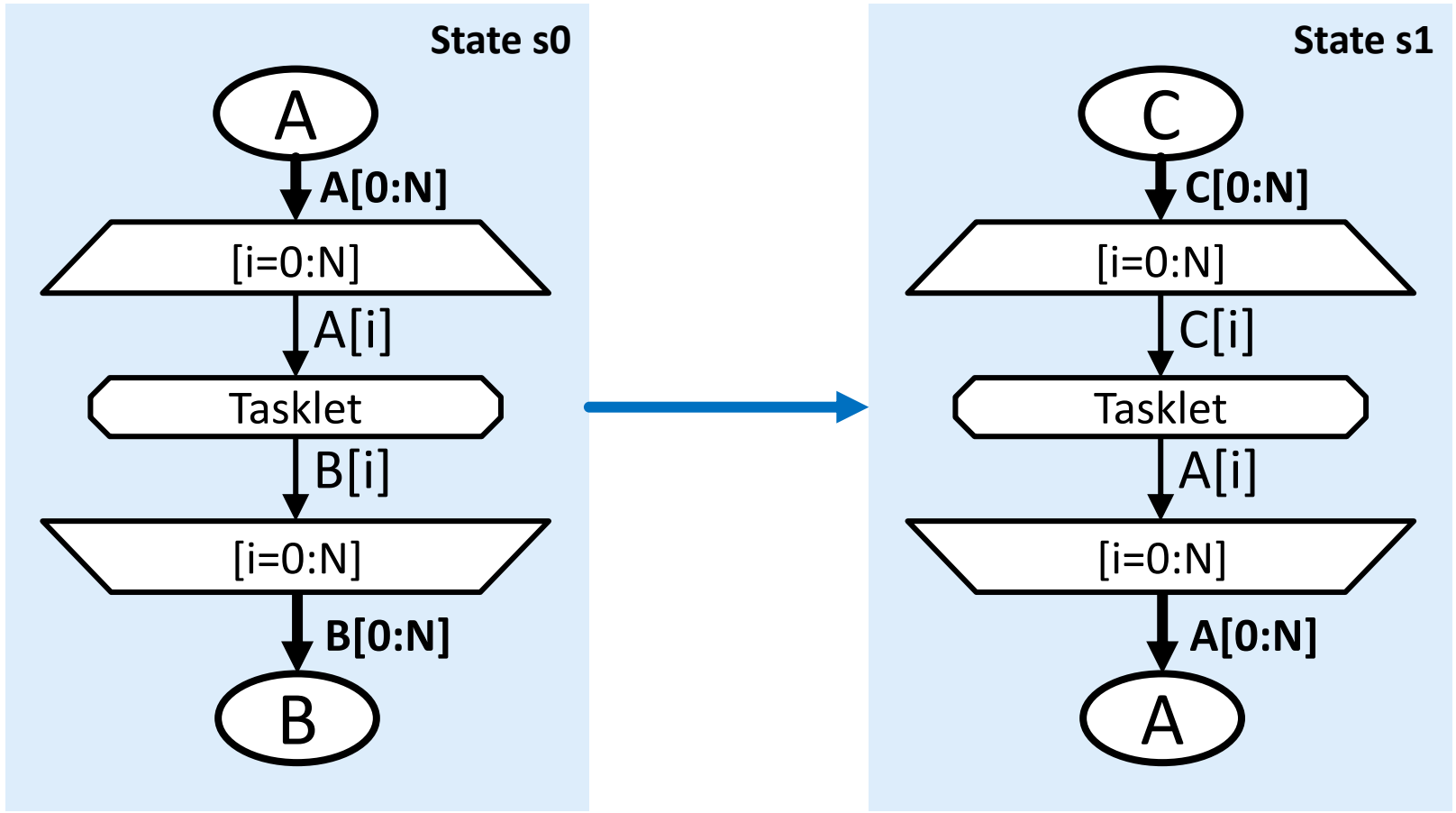


Stateful Parallel Dataflow Programming

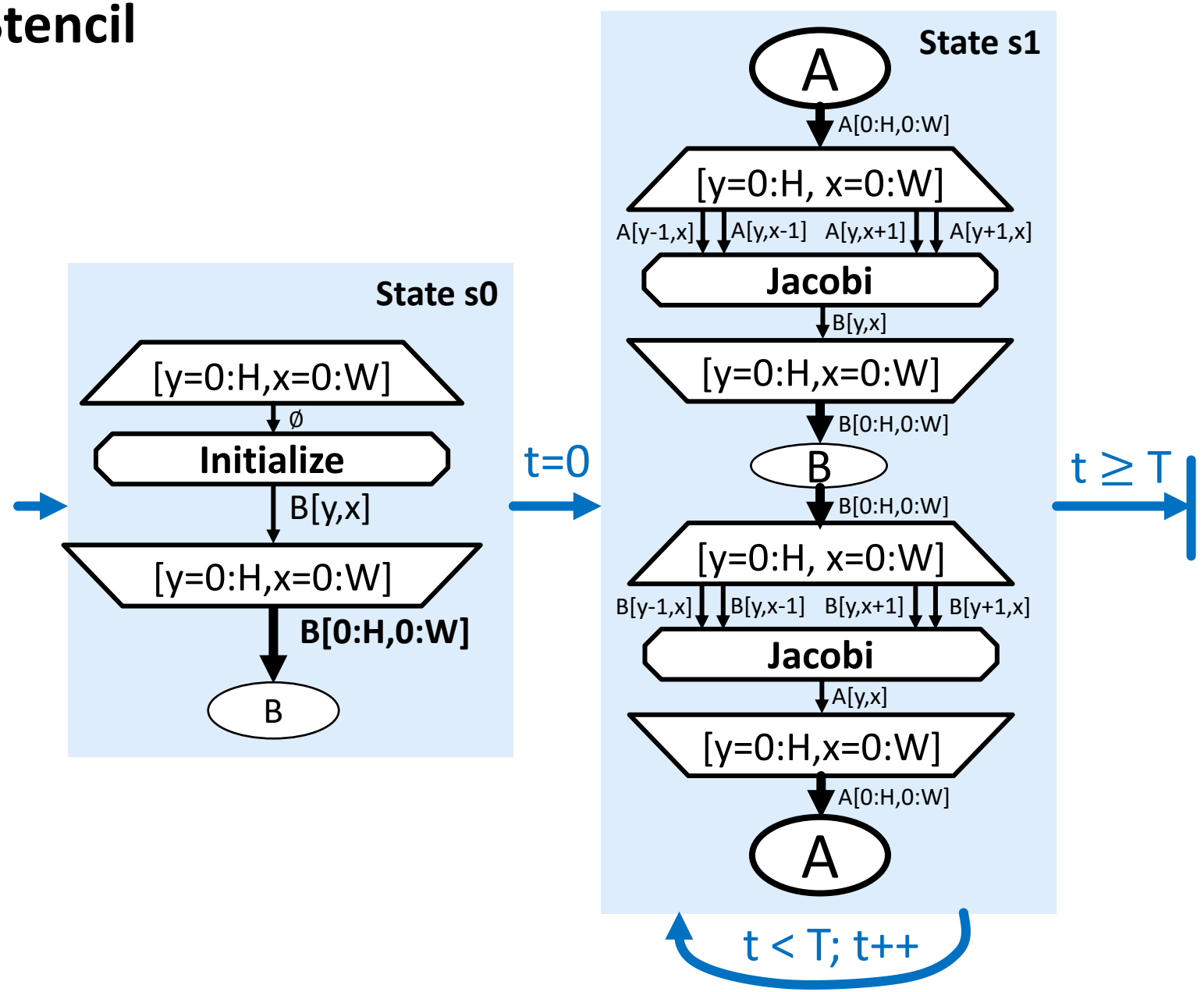




Stateful Parallel Dataflow Programming



Example: 2D Stencil



Meet the Nodes

State — State machine element

Tasklet — Fine-grained computational bl

Nested SDFG

Array — N-dimensional data container

Map — Parametric graph abstraction f

Exit

Stream — Streaming data container

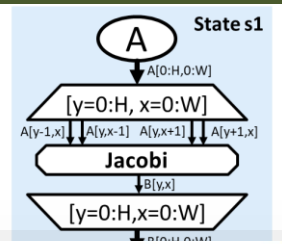
Consume — Dynamic mapping of computa

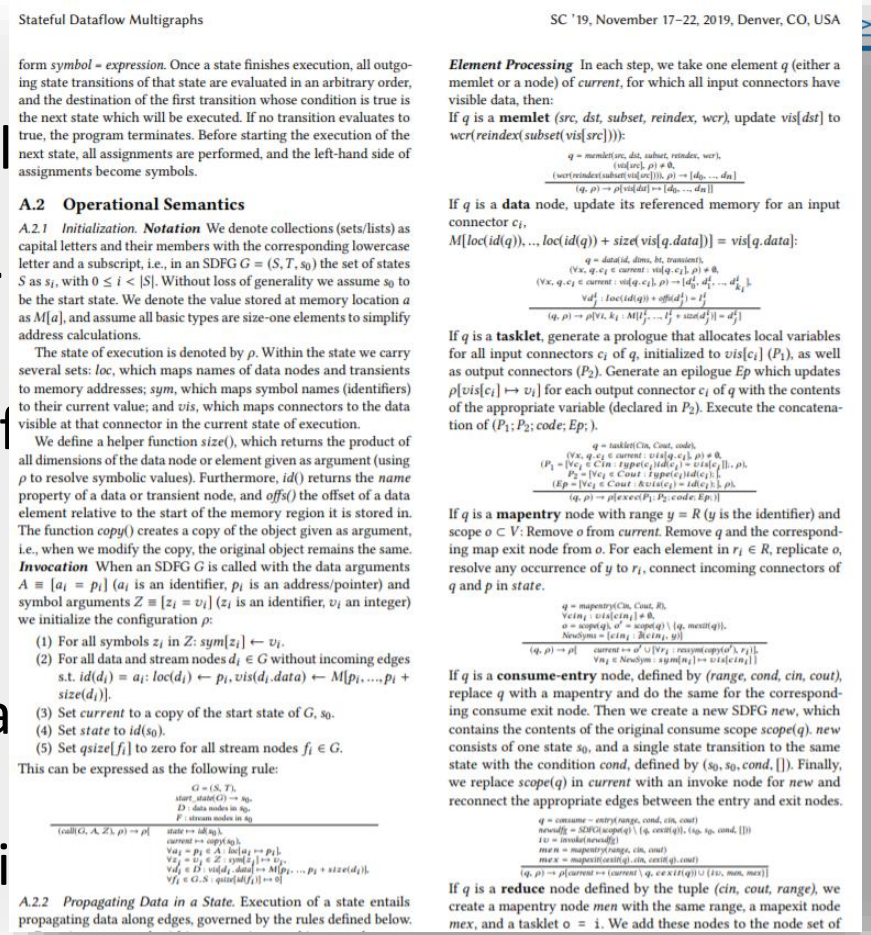
Exit

Conflict Resolution — Defines behavior during confli

Reduce

GEMM — Customizable computation with multiple implementations




Stateful Dataflow Multigraphs SC '19, November 17–22, 2019, Denver, CO, USA

form *symbol = expression*. Once a state finishes execution, all outgoing state transitions of that state are evaluated in an arbitrary order, and the destination of the first transition whose condition is true is the next state which will be executed. If no transition evaluates to true, the program terminates. Before starting the execution of the next state, all assignments are performed, and the left-hand side of assignments become symbols.

A.2 Operational Semantics

A.2.1 Initialization. ***Notation*** We denote collections (sets/lists) as capital letters and their members with the corresponding lowercase letter and a subscript, i.e., in an SDFG $G = (S, T, s_0)$ the set of states S as s_i, with $0 \leq i < |S|$. Without loss of generality we assume s_0 to be the start state. We denote the value stored at memory location a as $M[a]$, and assume all basic types are size-one elements to simplify address calculations.

The state of execution is denoted by ρ. Within the state we carry several sets: *loc*, which maps names of data nodes and transients to memory addresses; *sym*, which maps symbol names (identifiers) to their current value; and *vis*, which maps connectors to the data visible at that connector in the current state of execution.

We define a helper function *size*(), which returns the product of all dimensions of the data node or element given as argument (using ρ to resolve symbolic values). Furthermore, *id*() returns the *name* property of a data or transient node, and *offs*() the offset of a data element relative to the start of the memory region it is stored in. The function *copy*() creates a copy of the object given as argument, i.e., when we modify the copy, the original object remains the same.

Invocation When an SDFG G is called with the data arguments $A \equiv [a_i = p_i]$ (a_i is an identifier, p_i is an address/pointer) and symbol arguments $Z \equiv [z_i = v_i]$ (z_i is an identifier, v_i an integer) we initialize the configuration ρ:

(1) For all symbols z_i in Z: $sym[z_i] \leftarrow v_i$.
(2) For all data and stream nodes $d_i \in G$ without incoming edges s.t. $id(d_i) = a_i$: $loc(d_i) \leftarrow p_i, vis(d_i.data) \leftarrow M[p_i, \ldots, p_i + size(d_i)]$.
(3) Set *current* to a copy of the start state of G, s_0.
(4) Set *state* to $id(s_0)$.
(5) Set $qsize[f_i]$ to zero for all stream nodes $f_i \in G$.

This can be expressed as the following rule:

$$\frac{\begin{array}{l} G = (S, T), \\ start\_state(G) \rightarrow s_0, \\ D : \text{data nodes in } s_0, \\ F : \text{stream nodes in } s_0 \end{array}}{\begin{array}{l} (call(G, A, Z), \rho) \rightarrow \rho[\quad state \mapsto id(s_0), \\ current \mapsto copy(s_0), \\ \forall a_i = p_i \in A : loc[a_i \mapsto p_i], \\ \forall z_i = v_i \in Z : sym[z_i] \mapsto v_i, \\ \forall d_i \in D : vis[d_i.data] \mapsto M[p_i, \ldots, p_i + size(d_i)], \\ \forall f_i \in G.S : qsize[id(f_i)] \mapsto 0] \end{array}}$$

A.2.2 Propagating Data in a State. Execution of a state entails propagating data along edges, governed by the rules defined below.

Element Processing In each step, we take one element q (either a memlet or a node) of *current*, for which all input connectors have visible data, then:

If q is a **memlet** *(src, dst, subset, reindex, wcr)*, update $vis[dst]$ to $wcr(reindex(subset(vis[src])))$:

$$\frac{q = memlet(src, dst, subset, reindex, wcr), \quad (vis[src], \rho) \neq \emptyset, \quad (wcr(reindex(subset(vis[src])), \rho) \rightarrow [d_0, \ldots, d_n]}{(q, \rho) \rightarrow \rho[vis[dst] \mapsto [d_0, \ldots, d_n]]}$$

If q is a **data** node, update its referenced memory for an input connector c_i,
$M[loc(id(q)), .., loc(id(q)) + size(vis[q.data])] = vis[q.data]$:

$$\frac{\begin{array}{l} q = data(id, dims, bt, transient), \\ (\forall x, q.c_i \in current : vis[q.c_i], \rho) \neq \emptyset, \\ (\forall x, q.c_i \in current : vis[q.c_i], \rho) \rightarrow [d_0^i, d_1^i, \ldots, d_{k_i}^i], \\ \forall d_j^i : loc(id(q)) + offs(d_j^i) = l_j^i \end{array}}{(q, \rho) \rightarrow \rho[\forall i, k_i : M[l_j^i, \ldots, l_j^i + size(d_j^i)] = d_j^i]}$$

If q is a **tasklet**, generate a prologue that allocates local variables for all input connectors c_i of q, initialized to $vis[c_i]$ (P_1), as well as output connectors (P_2). Generate an epilogue Ep which updates $\rho[vis[c_i] \mapsto v_i]$ for each output connector c_i of q with the contents of the appropriate variable (declared in P_2). Execute the concatenation of (P_1; P_2; *code*; Ep;).

$$\frac{\begin{array}{l} q = tasklet(Cin, Cout, code), \\ (\forall x, q.c_i \in current : vis[q.c_i], \rho) \neq \emptyset, \\ (P_1 = [\forall c_i \in Cin : type(c_i)id(c_i) \mapsto vis[c_i]], \rho), \\ P_2 = [\forall c_i \in Cout : type(c_i)id(c_i)], \\ (Ep = [\forall c_i \in Cout : \&vis(c_i) \mapsto id(c_i)], \rho), \end{array}}{(q, \rho) \rightarrow \rho[exec(P_1; P_2; code; Ep;)]}$$

If q is a **mapentry** node with range $y = R$ (y is the identifier) and scope $o \subset V$: Remove o from *current*. Remove q and the corresponding map exit node from o. For each element in $r_i \in R$, replicate o, resolve any occurrence of y to r_i, connect incoming connectors of q and p in *state*.

$$\frac{\begin{array}{l} q = mapentry(Cin, Cout, R), \\ \forall cin_i : vis[cin_i] \neq \emptyset, \\ o = scope(q), o' = scope(q) \setminus \{q, mexit(q)\}, \\ Newsyms = [cin_i : R(cin_i, y)] \end{array}}{\begin{array}{l} (q, \rho) \rightarrow \rho[\quad current \mapsto o' \cup [\forall r_i : reasymt.copy(o'), r_i)], \\ \forall n_i \in Newsym : sym[n_i] \mapsto vis[cin_i]] \end{array}}$$

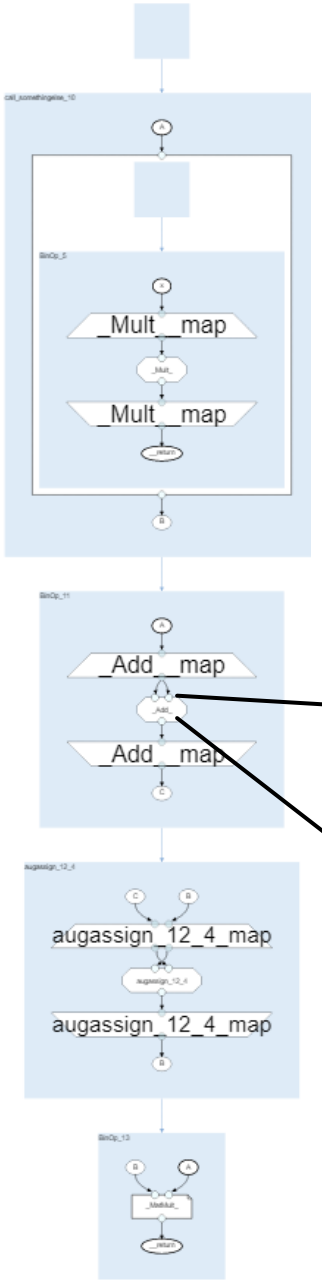
If q is a **consume-entry** node, defined by *(range, cond, cin, cout)*, replace q with a mapentry and do the same for the corresponding consume exit node. Then we create a new SDFG *new*, which contains the contents of the original consume scope *scope*(q). *new* consists of one state s_0, and a single state transition to the same state with the condition *cond*, defined by (s_0, s_0, *cond*, [ ]). Finally, we replace *scope*(q) in *current* with an invoke node for *new* and reconnect the appropriate edges between the entry and exit nodes.

$$\frac{\begin{array}{l} q = consume-entry(range, cond, cin, cout) \\ newsdfg = SDFG(scope(q) \setminus \{q, cexit(q)\}, (s_0, s_0, cond, [])) \\ ic = invoke(newsdfg) \\ men = mapentry(range, cin, cout) \\ mex = mapexit(cexit(q), cin, cexit(q).cout) \end{array}}{(q, \rho) \rightarrow \rho[current \mapsto (current \setminus q, cexit(q)) \cup (ic, men, mex)]}$$

If q is a **reduce** node defined by the tuple *(cin, cout, range)*, we create a mapentry node *men* with the same range, a mapexit node *mex*, and a tasklet o = i. We add these nodes to the node set of

SDFG <-> MLIR

- **Symbols vs. Identifiers**
  - Different from scalars
- **Tasklets vs. Basic Blocks**
  - Granularity does not matter
  - Data movement does



SDFG

```
A: float64[N]
B: float64[N + 1]

C = A + B
D = A + B[1:]
```

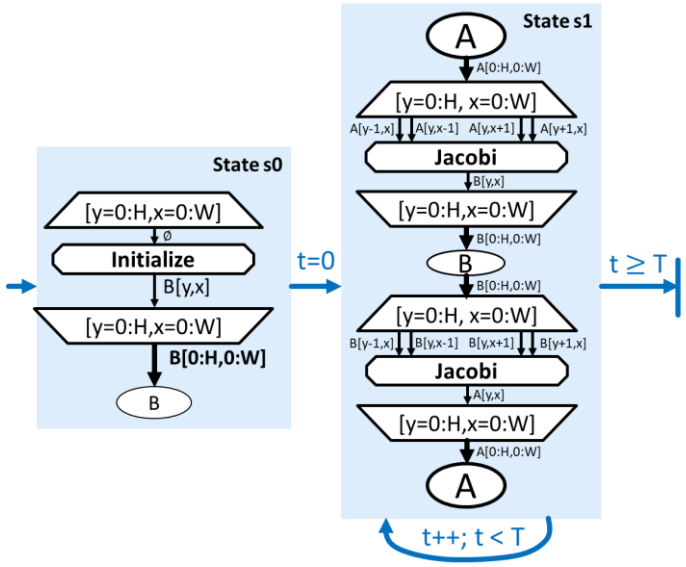
MLIR

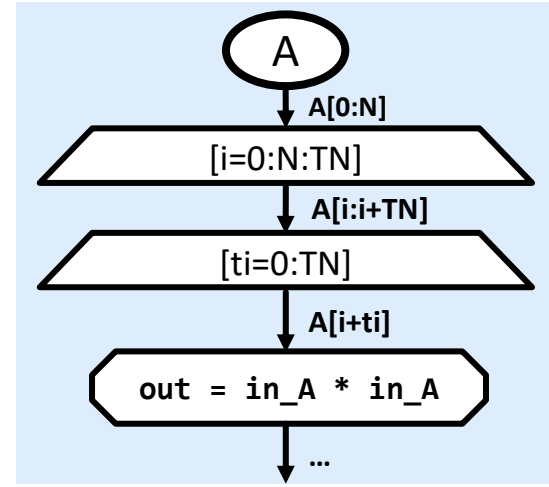
```
A: memref<?xf64>
B: memref<?xf64>

%C = linalg.add %A, %B
```

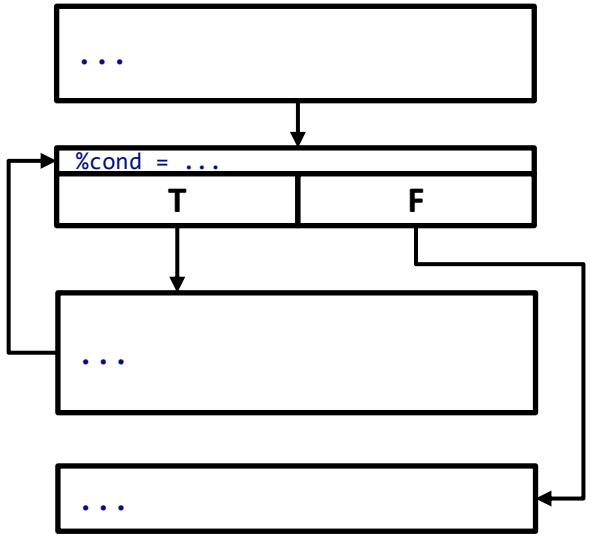
```
x0 = -2.5 + ((float(px) / W) * 3.5)
y0 = -1 + ((float(py) / H) * 2)
x = 0.0
y = 0.0
iteration = 0
while (x * x + y * y < 2 * 2 and iteration < MAXITER):
    xtemp = x * x - y * y + x0
    y = 2 * x * y + y0
    x = xtemp
    iteration = iteration + 1

out = iteration
```

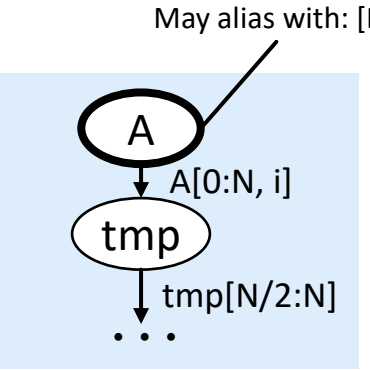
SDFG <-> MLIR

- **Symbols vs. Identifiers**
  - Different from scalars
- **Tasklets vs. Basic Blocks**
  - Granularity does not matter
  - Data movement does
- **Symbolic state machine vs. CFG**
  - Importance of data dependence on state transition
- **Memlets vs. Memrefs**
  - Always explicit, absolute
  - Can be traced to concrete memory locations
- **Transient containers vs. alloc(a)**
  - Dead Memory Elimination
  - Aliasing not a concern (most of the time…)









DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

Python | DSLs

PyTorch | MATLAB

...

Scientific Frontend

```
glob_b = ...
class ClassA:
    def __init__(self, arr):
        self.q = arr

    @dace.method
    def __call__(self, a):
        return a * self.q + glob_b
```

JIT

```
@dace_module
class Model(nn.Module):
    def __init__(self, kernel_size):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 4, kernel_size)
        self.conv2 = nn.Conv2d(4, 4, kernel_size)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

```
N = dace.symbol()

@dace.program
def jacobi_1d(tsteps: dace.int32,
              a: dace.float64[N],
              b: dace.float64[N]):

    for _ in range (1, tsteps):
        b[1: -1] = 0.33333 * (
            a[:-2] + a[1:-1] + a[2:])
        a[1: -1] = 0.33333 * (
            b[:-2] + b[1:-1] + b[2:])
```

AOT

```
for (int i = 0; i < N ; i++)
    for (int j = row_ptr[i]; j < row_ptr[i+1]; j++)
        y[i] += A[col_idx[j]] * x[j];
```

DaCe Overview

Domain Scientist

Problem Formulation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

Python | DSLs

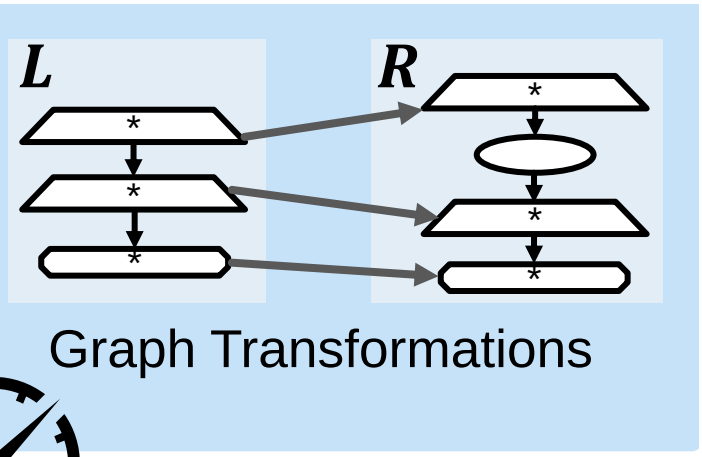
PyTorch | MATLAB

...

Scientific Frontend

Performance Engineer

Data-Centric Intermediate Representation (SDFG)



Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

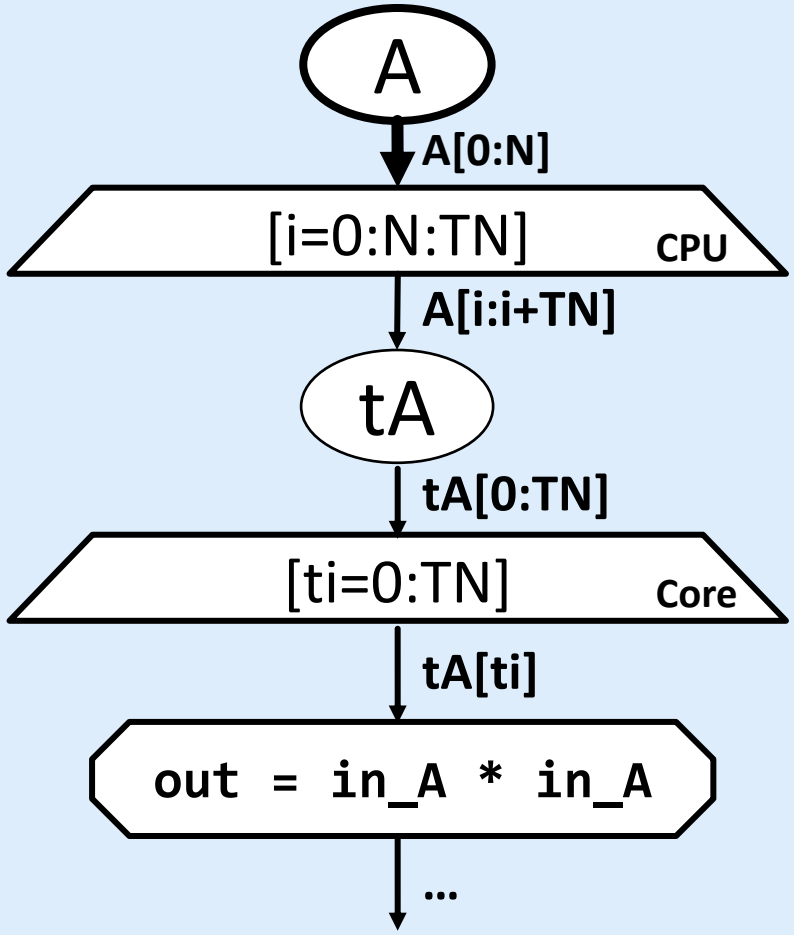
CPU Binary

GPU Binary

FPGA Modules

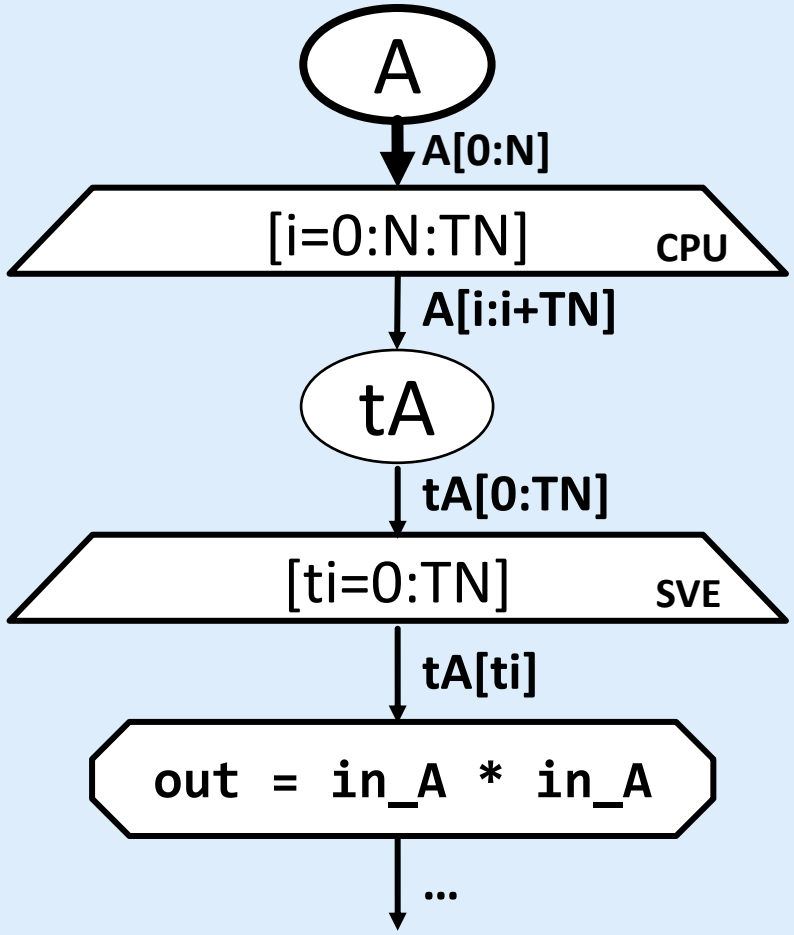
Hierarchical Parallelism and Heterogeneity

- **Maps have schedules, arrays have storage locations**



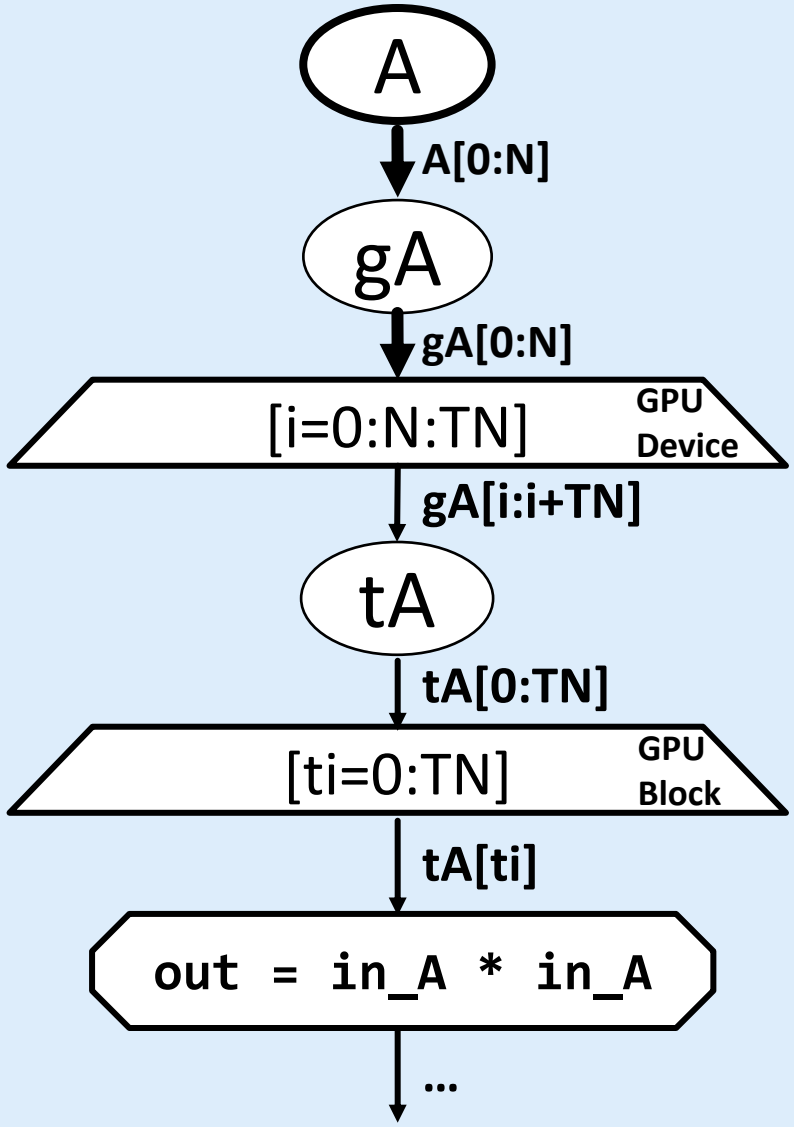

```
// ...
#pragma omp parallel for
for (int i = 0; i < N; i += TN) {
    vec<double, 4> tA[TN];
    Global2Stack_1D<double, 4, 1>(&A[i], min(N - i, TN), tA);

    for (int ti = 0; ti < TN; ti += 1) {
        vec<double, 4> in_A = tA[ti];
        auto out = (in_A * in_A);
        tC[ti] = out;
    }
```

Hierarchical Parallelism and Heterogeneity




```
// ...
#pragma omp parallel for
for (int i = 0; i < N; i += TN) {
  // ...
  int64_t ti = 0;
  svbool_t pg = svwhilelt_b64(ti, TN);
  do {
      svfloat64_t in_A = svld1(pg, &tA[ti]);
      svst1(pg, &tC[ti],
            svmul_x(pg, in_A, in_A));
      ti += svcntd();
      pg = svwhilelt_b64(ti, TN);
  }
  while (svptest_any(svptrue_b64(), pg));
}
```

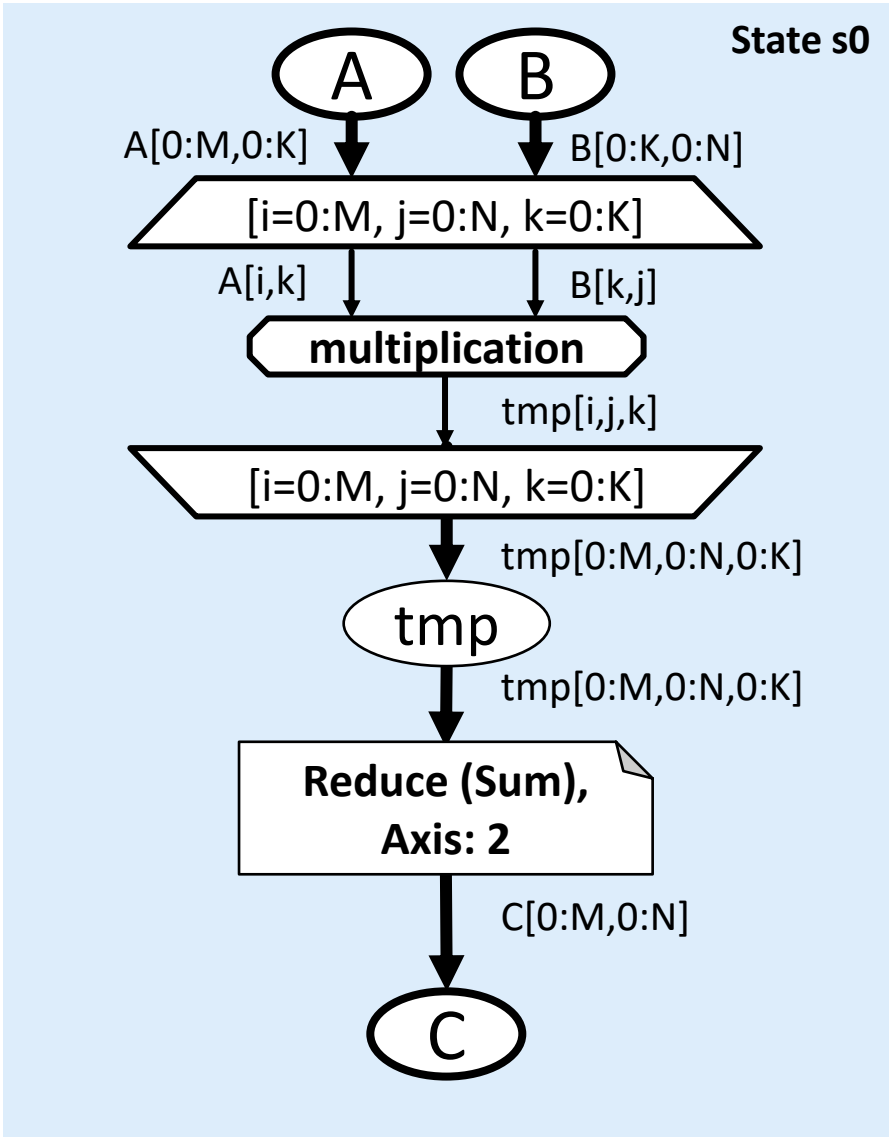
Hierarchical Parallelism and Heterogeneity




```
__global__ void multiplication_1(...) {
    int i = blockIdx.x * TN;
    int ti = threadIdx.y + 0;
    if (i+ti >= N)  return;

    __shared__ vec<double, 2> tA[TN];
    GlobalToShared1D<double, 2, TN, 1, 1, false>(gA, tA);

    vec<double, 2> in_A = tA[ti];
    auto out = (in_A * in_A);
    tC[ti] = out;
}
```

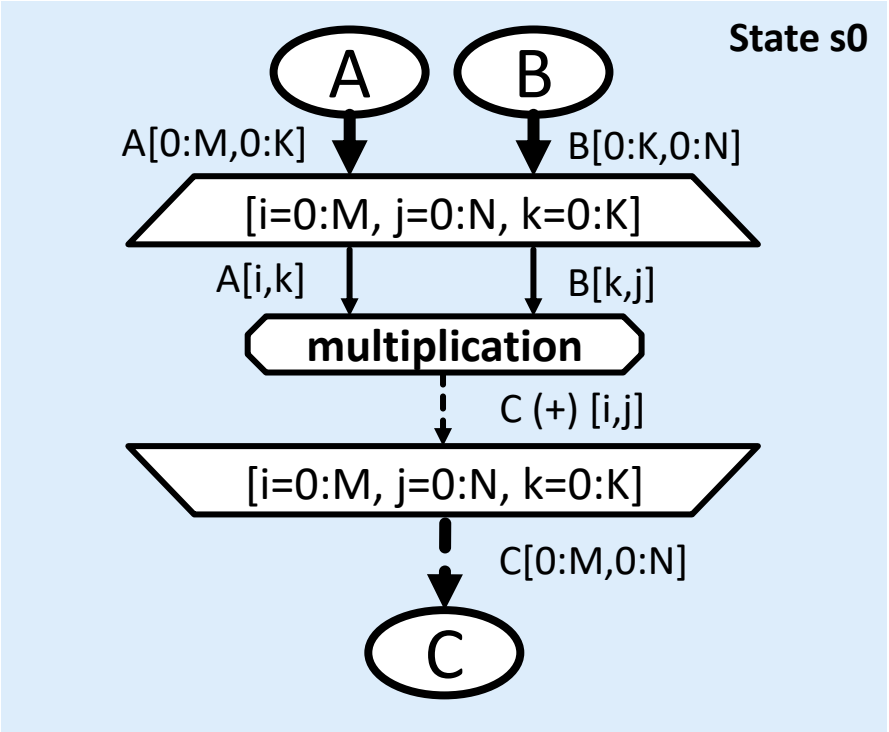
Matrix Multiplication SDFG

```
@dace.program
def gemm(A: dace.float64[M, K], B: dace.float64[K, N],
         C: dace.float64[M, N]):
    # Transient variable
    tmp = np.ndarray([M, N, K], dtype=A.dtype)

    for i, j, k in dace.map[0:M, 0:N, 0:K]:
        tmp[i, j, k] = A[i, k] * B[k, j]

    C[:] = numpy.sum(tmp, axis=2)
```

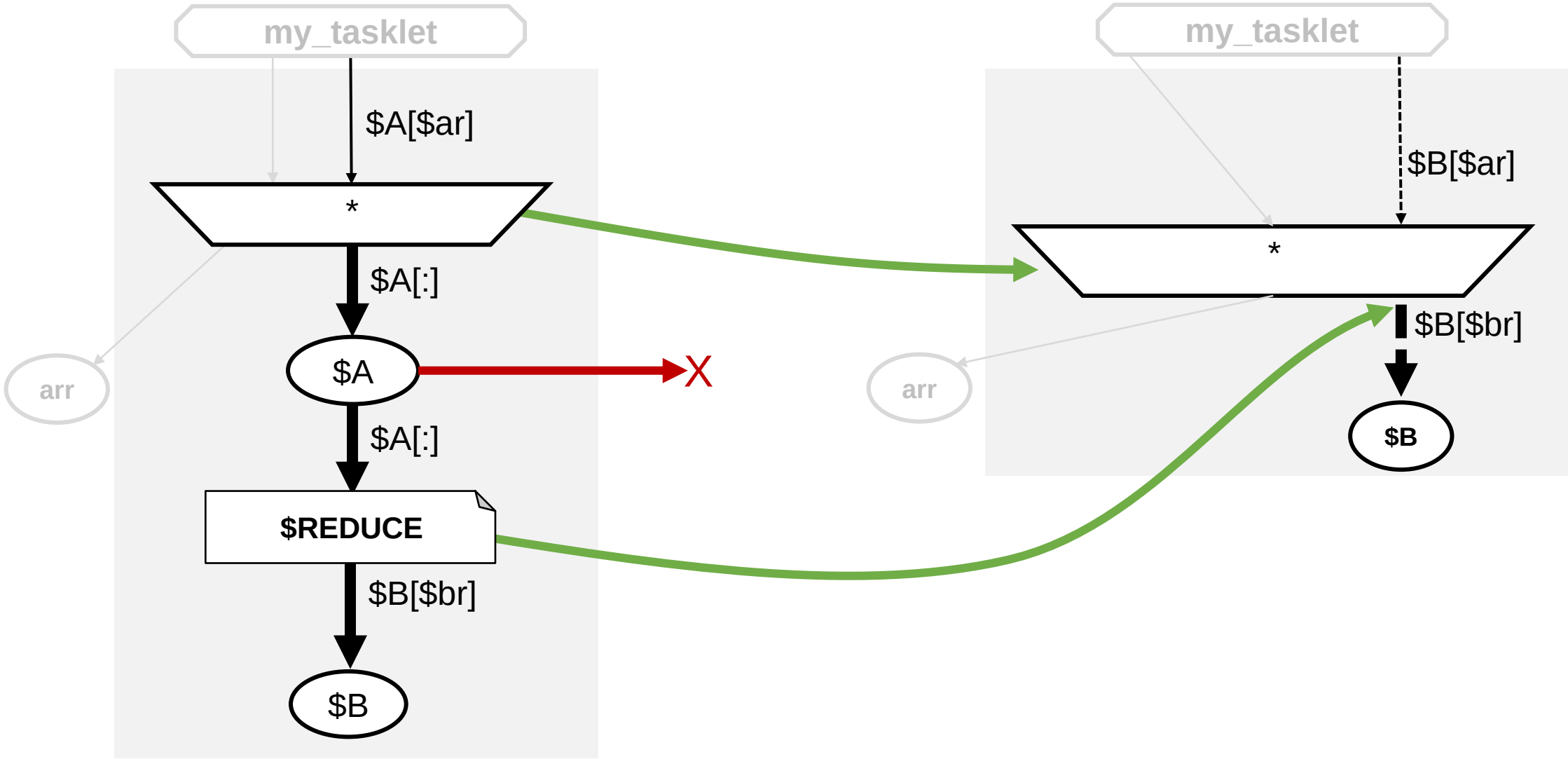


Matrix Multiplication SDFG

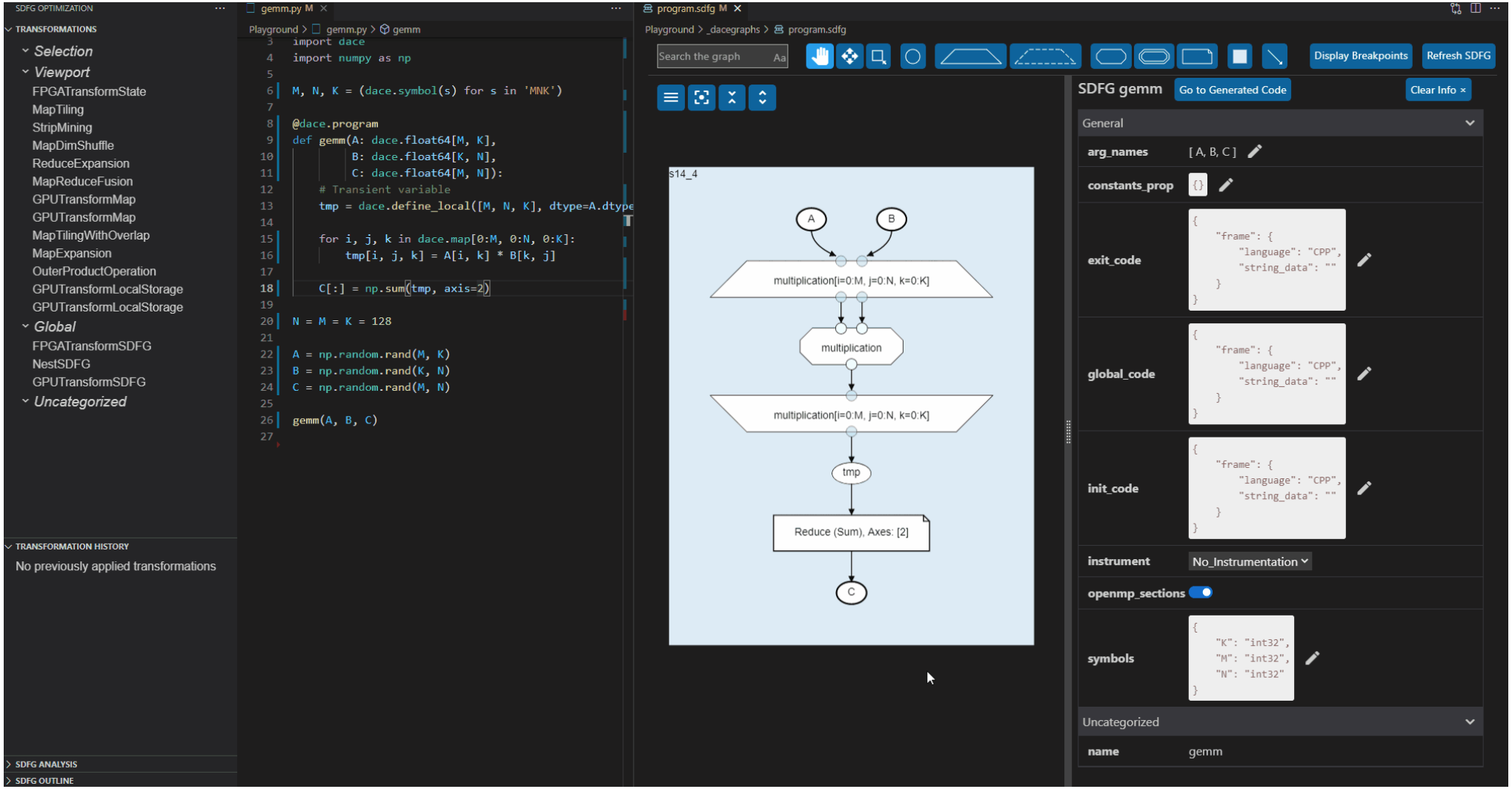
```
@dace.program
def gemm(A: dace.float64[M, K], B: dace.float64[K, N],
         C: dace.float64[M, N]):
    # Transient variable
    tmp = np.ndarray([M, N, K], dtype=A.dtype)

    for i, j, k in dace.map[0:M, 0:N, 0:K]:
        tmp[i, j, k] = A[i, k] * B[k, j]

    C[:] = numpy.sum(tmp, axis=2)
```



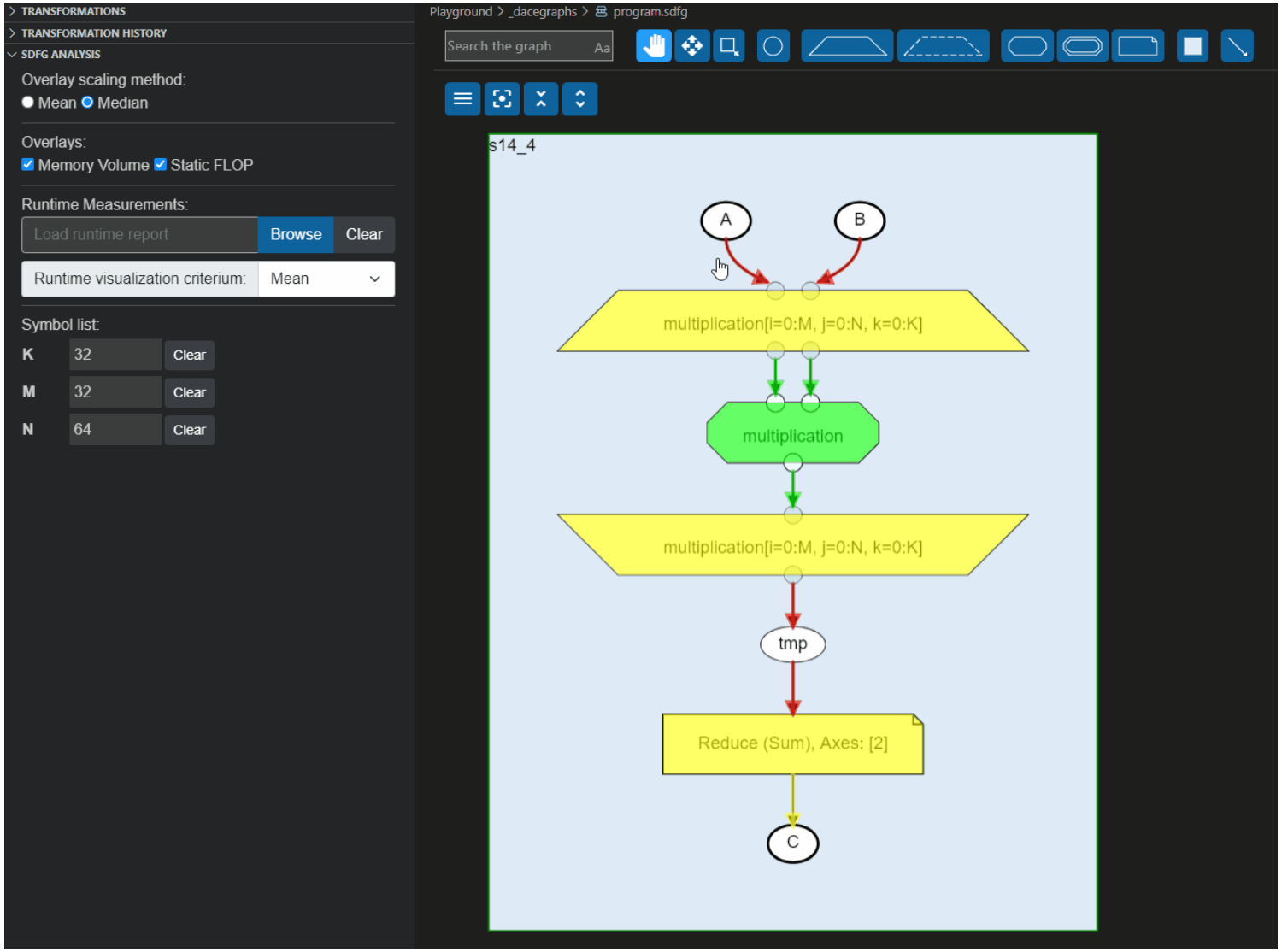
MapReduceFusion Transformation



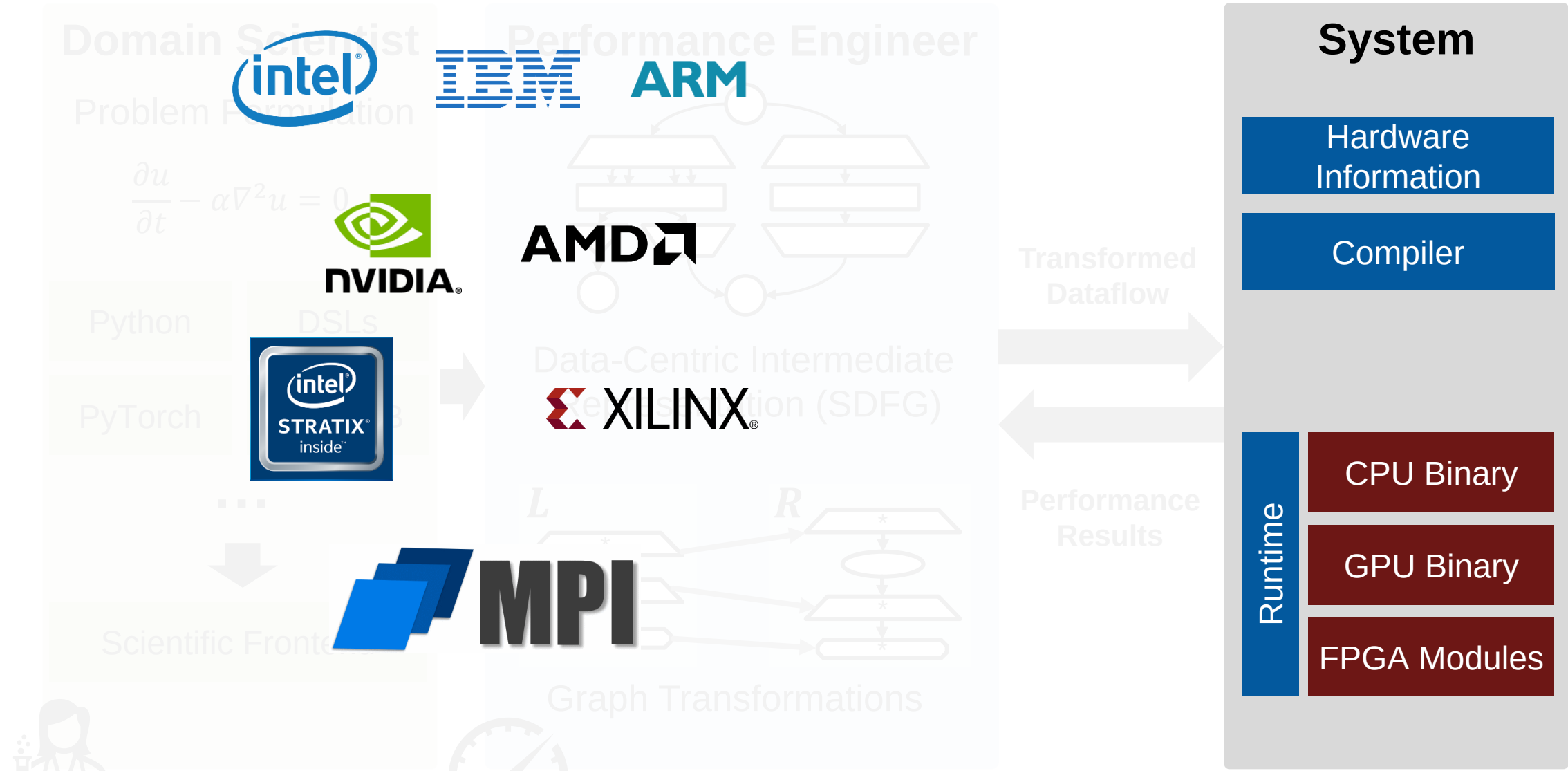
Visual Studio Code Integration



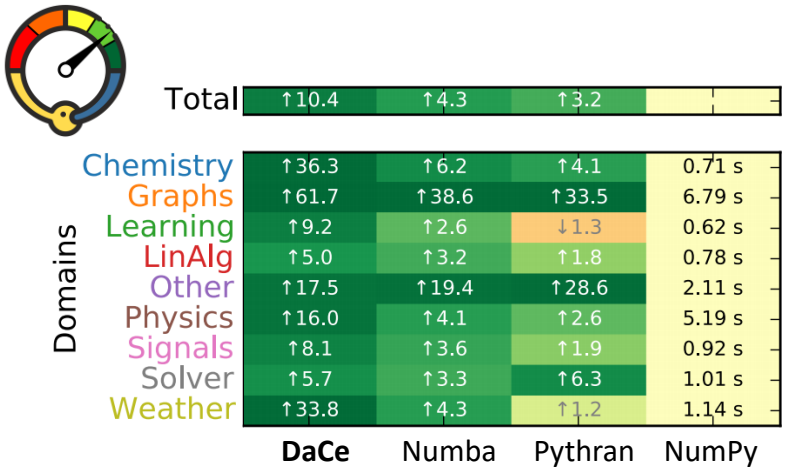
Visual Studio Code Integration



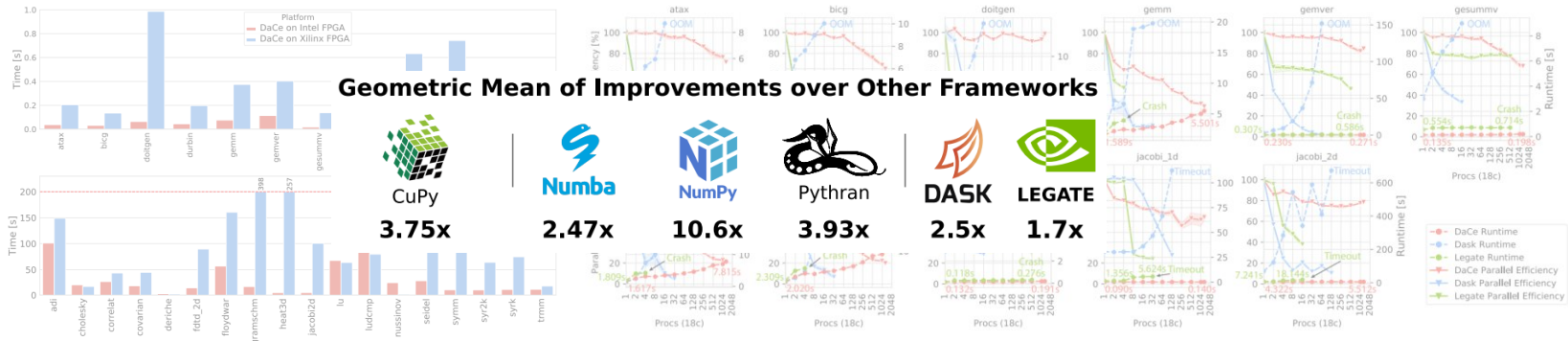
DaCe Overview



Performance

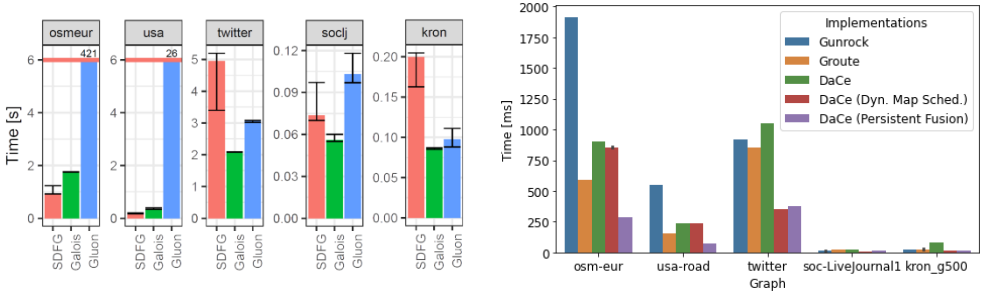


NumPyBench

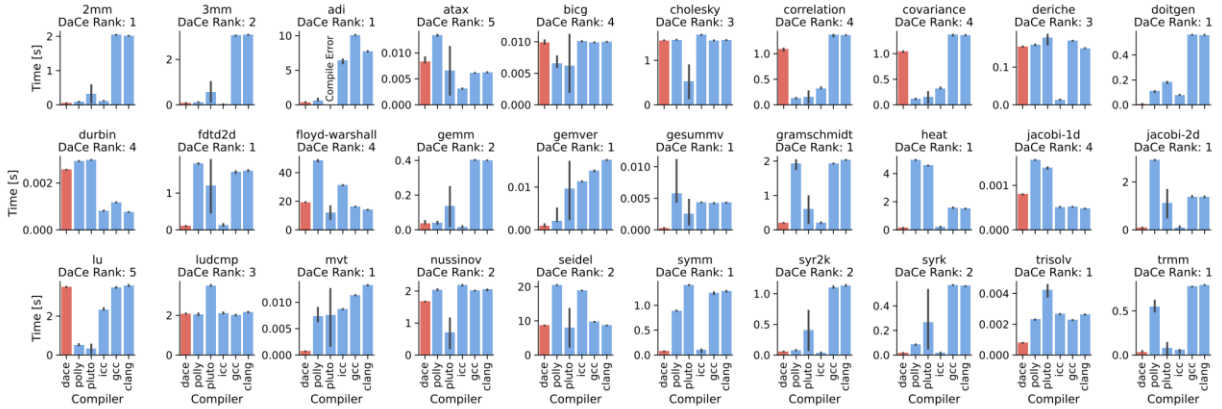


NumPyBench FPGA

NumPyBench Distributed



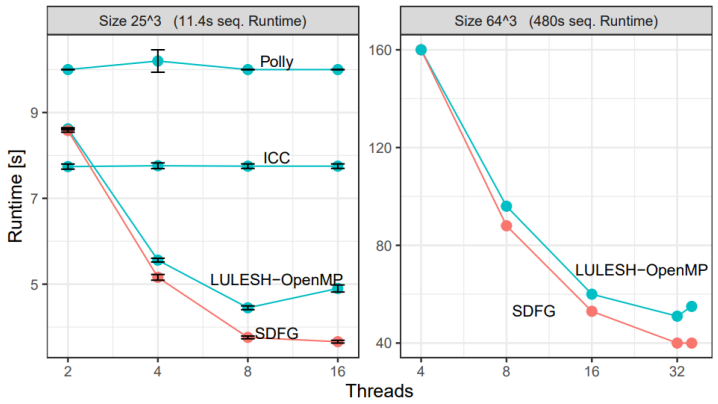
Graph Analytics

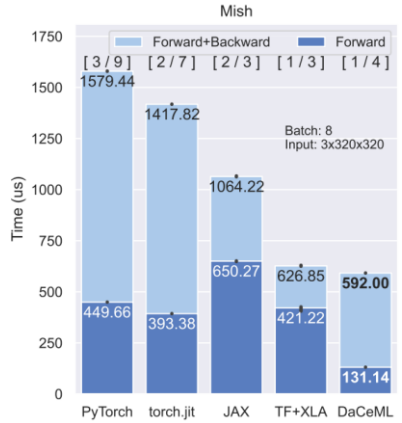


PolyBench/C

https://github.com/spcl/npbench

Performance

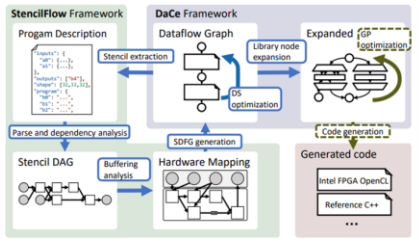


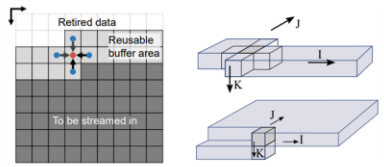


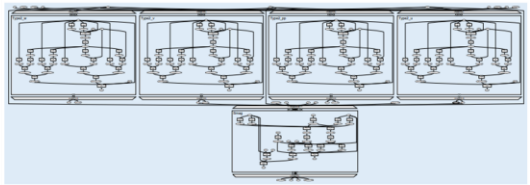
Unstructured Hydrodynamics (LULESH)

		PyTorch		torch.jit		JAX		TF+XLA		DaCeML	
		→	⇄	→	⇄	→	⇄	→	⇄	→	⇄
Guided	EfficientNet	2.05	6.90	2.04	6.94	2.39	7.40	1.54	6.37	**1.40**	**5.97**
	$BERT_{LARGE}$ (mixed)	2.94	8.18	2.92	8.20	3.19	8.11	3.80	10.76	**2.74**	**7.62**
Automatic	ResNet-50	14.55	32.04	**9.98**	**31.94**	14.17	33.93	12.33	35.57	10.03	32.45
	Wide ResNet-50-2	22.50	70.94	22.45	70.83	40.49	98.13	32.79	99.06	**20.62**	**67.99**
	MobileNet V2	9.98	18.45	6.22	15.53	—	—	7.42	20.29	**4.74**	**14.77**
	EfficientNet	2.05	6.90	2.04	6.94	2.39	7.40	**1.54**	**6.37**	1.57	15.00
	MLP Mixer	1.63	**3.65**	**1.36**	3.66	1.77	4.01	—	—	1.48	4.25
	FCN8s	46.85	158.42	46.82	**158.40**	—	—	—	—	**45.97**	166.30
	WaveNet	23.21	46.39	**18.67**	41.49	—	—	—	—	26.16	**41.07**
	$BERT_{LARGE}$ (single)	11.05	31.76	11.05	31.82	**10.93**	**29.94**	11.14	38.73	11.44	32.98
	$BERT_{LARGE}$ (mixed)	2.94	8.18	**2.92**	8.20	3.19	**8.11**	3.80	10.76	3.34	9.25
	DLRM	118.07	126.55	**117.38**	126.83	—	—	—	—	117.69	**126.42**

Deep Learning (DaCeML)



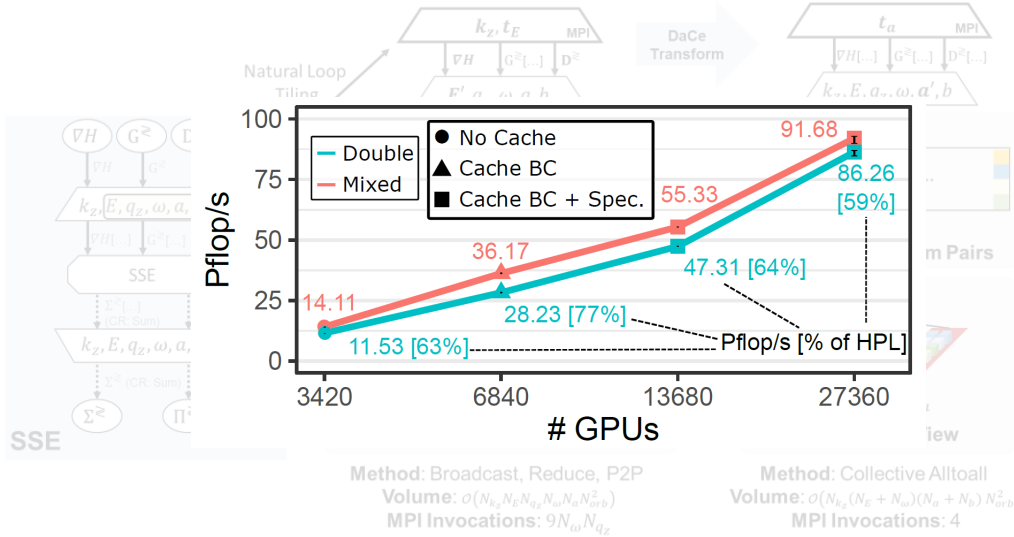




	Runtime	Performance	Peak BW.	%Roof.
Stratix 10	1,178 µs	145 GOp/s	77 GB/s	52%
Stratix 10*	332 µs	513 GOp/s	∞ GB/s	–
Xeon 12C	5,270 µs	32 GOp/s	68 GB/s	13%
P100	810 µs	210 GOp/s	732 GB/s	8%
V100	201 µs	849 GOp/s	900 GB/s	26%

*Without memory bandwidth constraints.

Numerical Weather Prediction (on CPUs, GPUs, and spatial architectures)



Quantum Transport Simulation

Berke Ates, Prof. Dr. Torsten Hoefler, Dr. Tal Ben-Nun, Dr. Alexandru Calotoiu

SDIR: A Data-Centric Dialect for MLIR

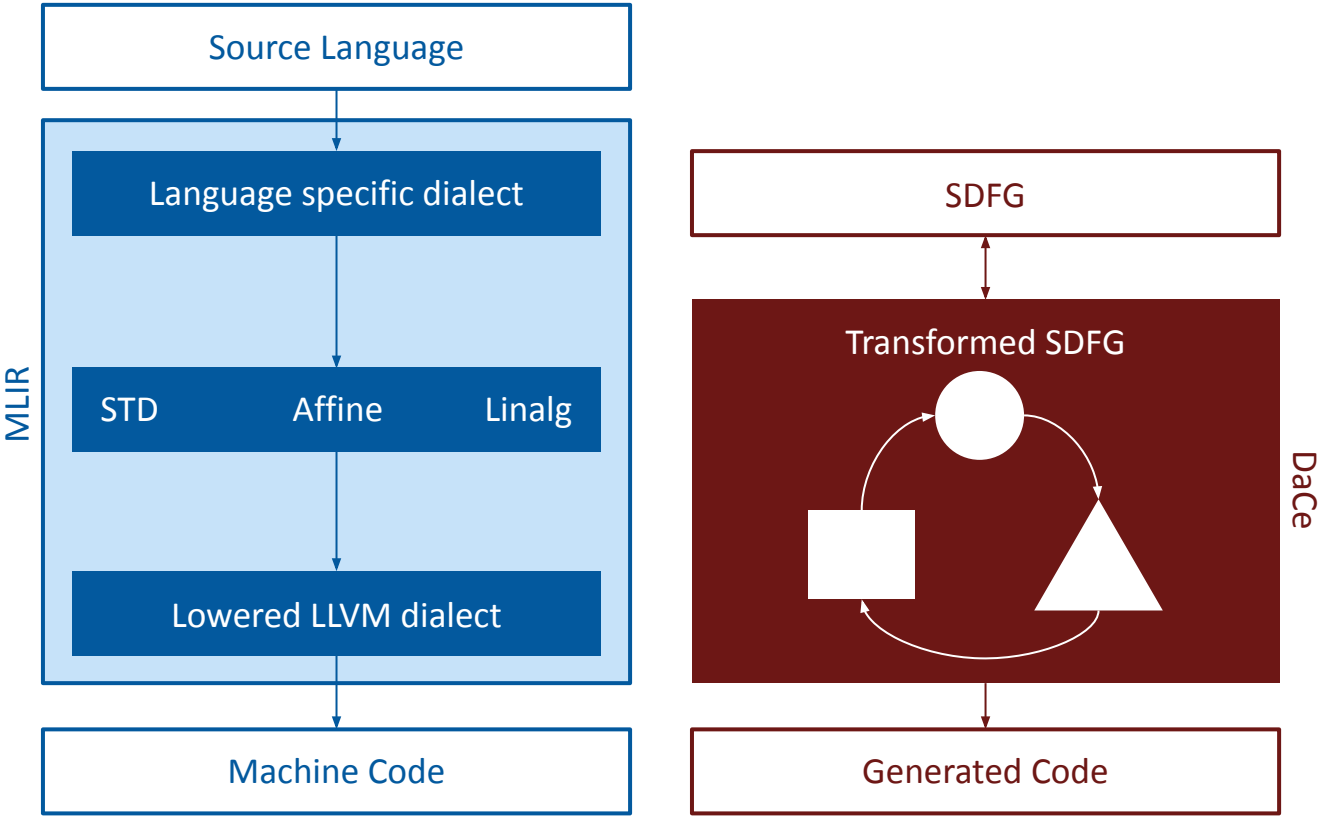
Outline

- Motivation
- SDIR
- Evaluation
- Summary

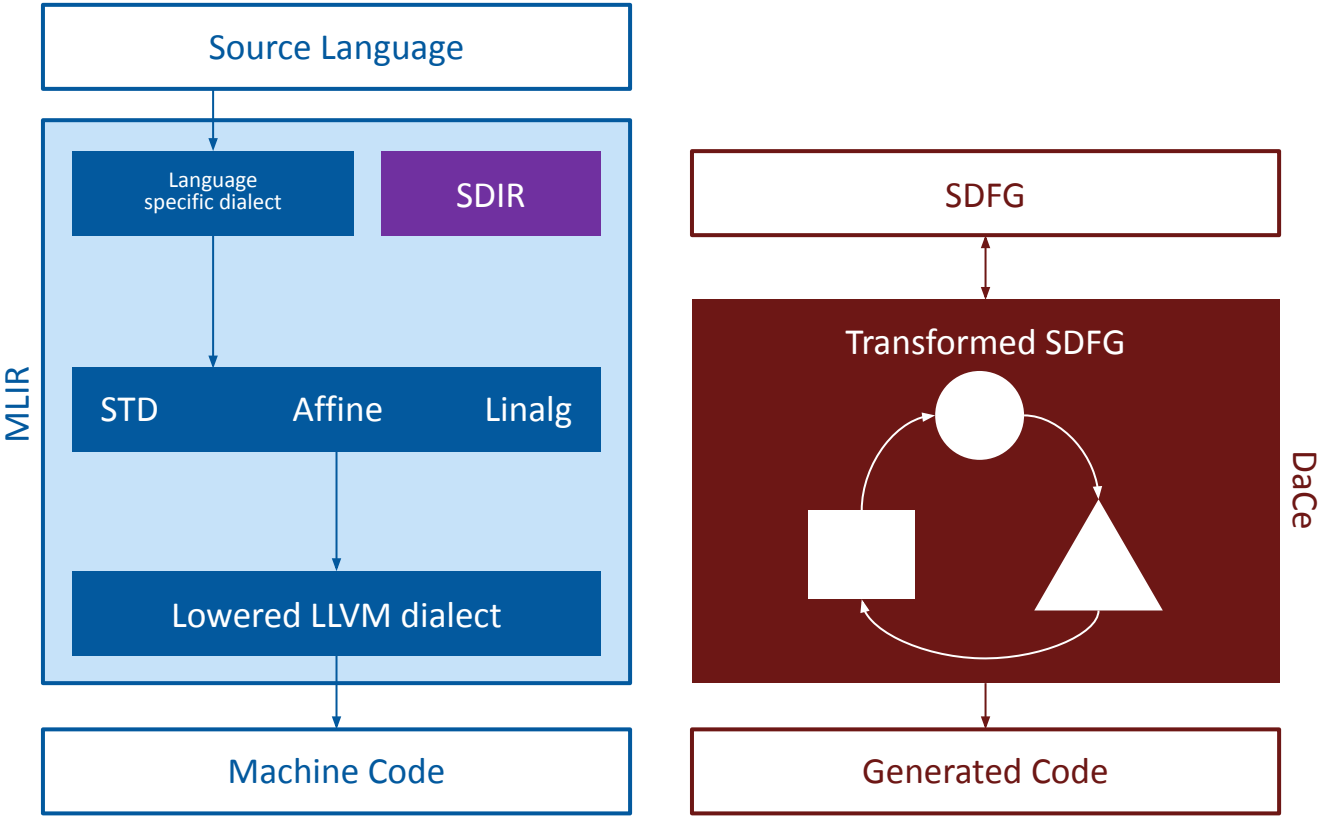
Outline

- **Motivation**
- SDIR
- Evaluation
- Summary

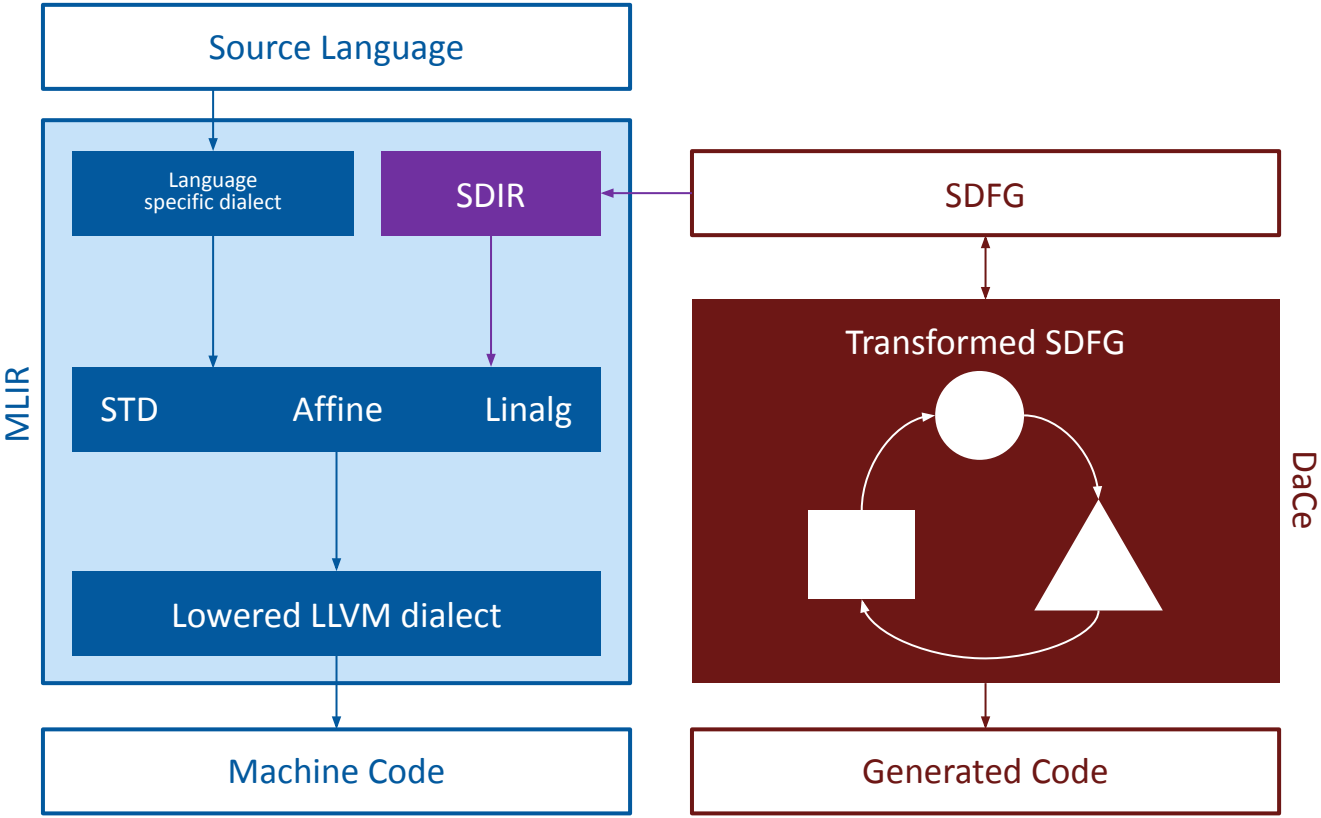
SDIR: Between MLIR and DaCe



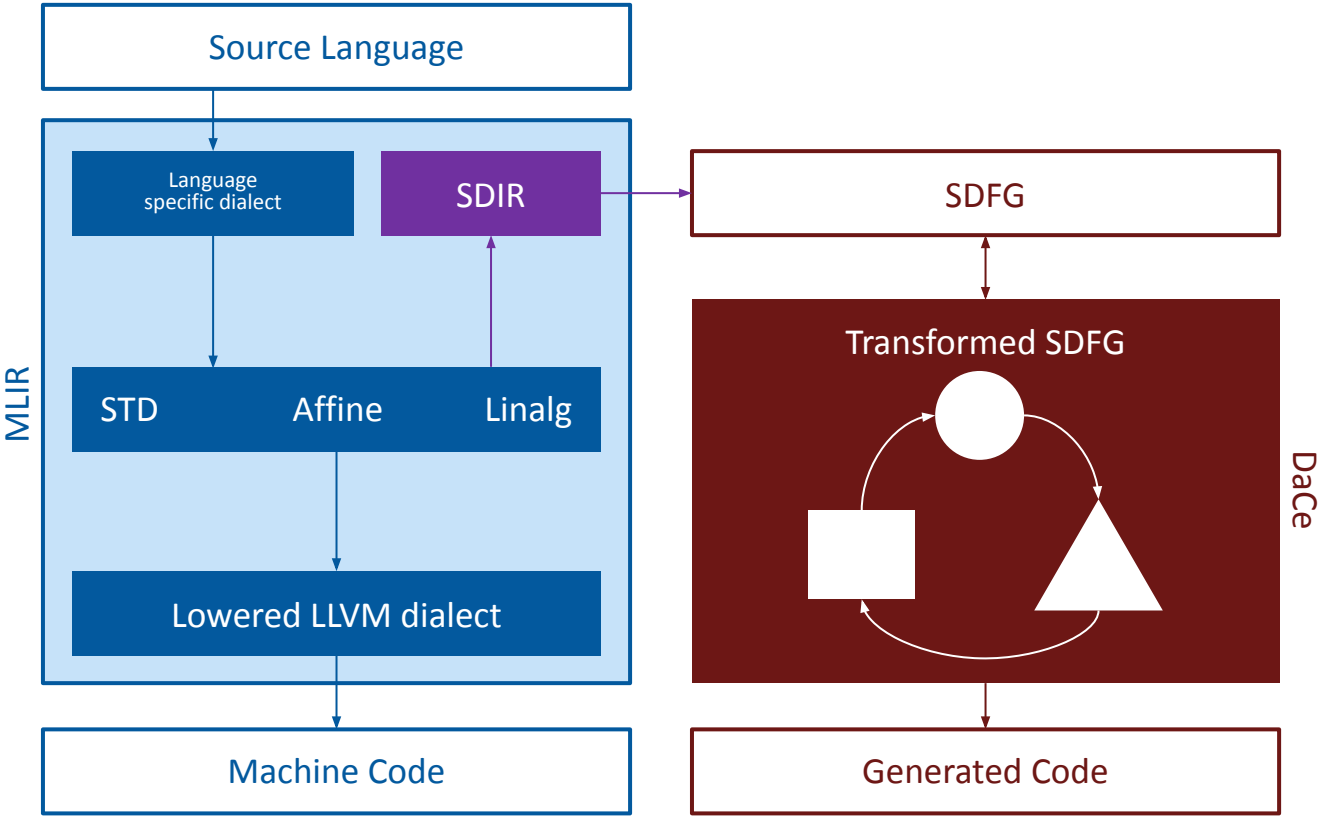
SDIR: Between MLIR and DaCe



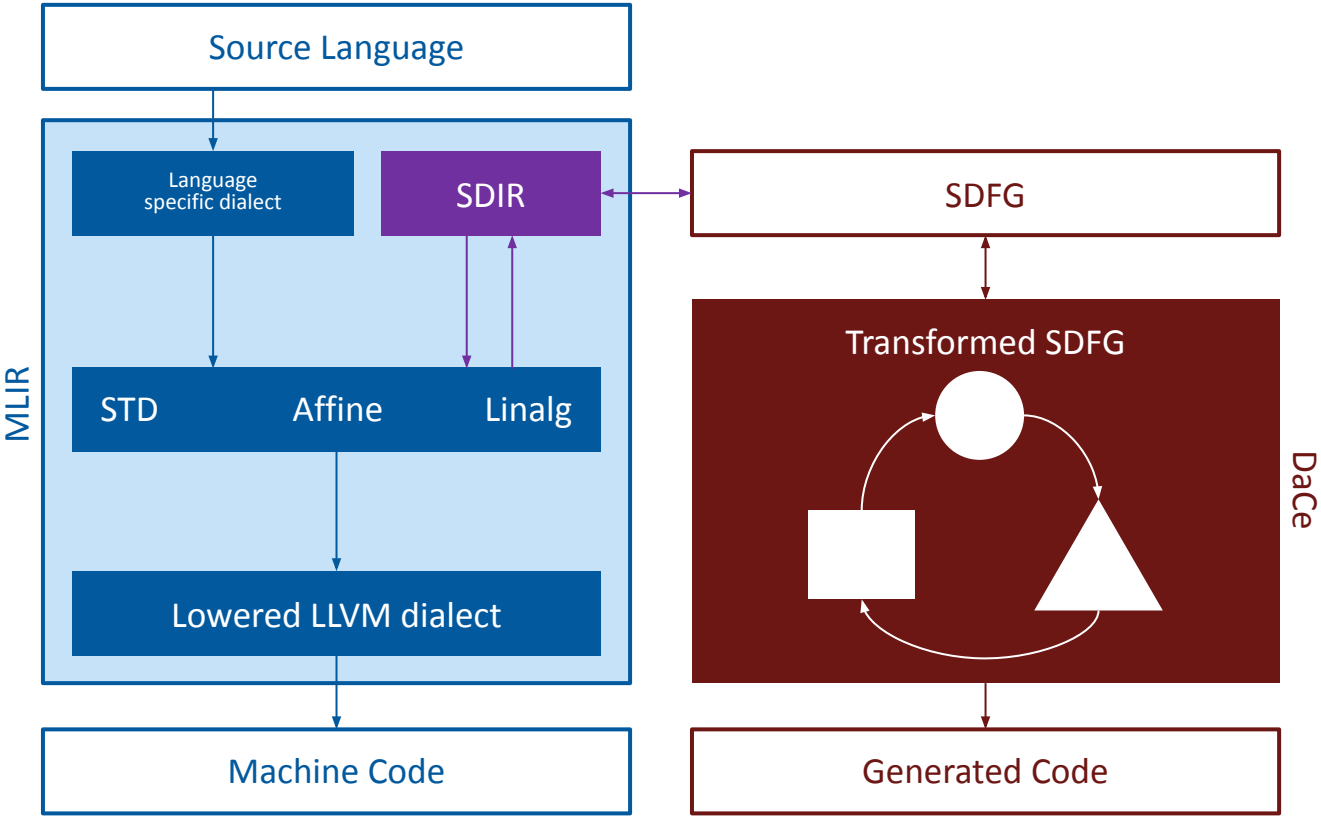
SDIR: Between MLIR and DaCe



SDIR: Between MLIR and DaCe



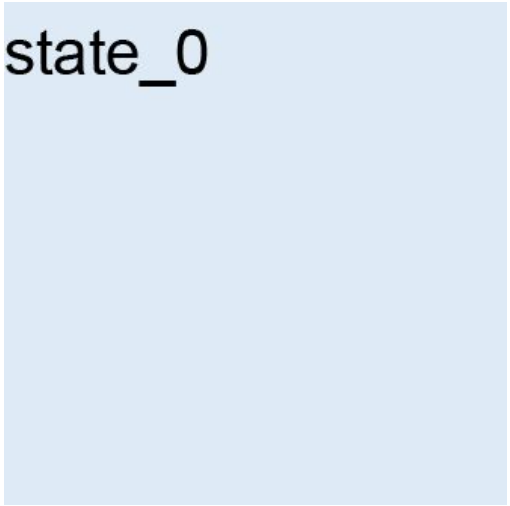
SDIR: Between MLIR and DaCe



Outline

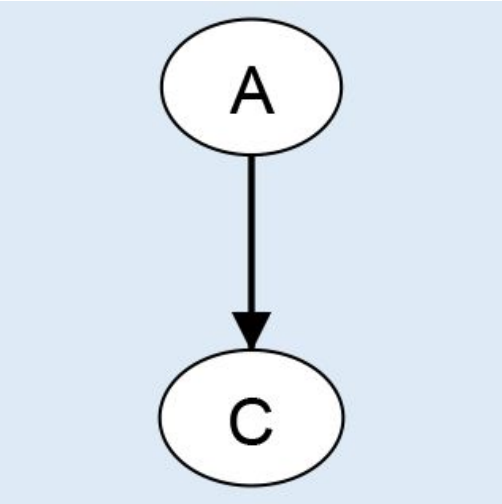
- Motivation
- **SDIR**
- Evaluation
- Summary

SDIR: States



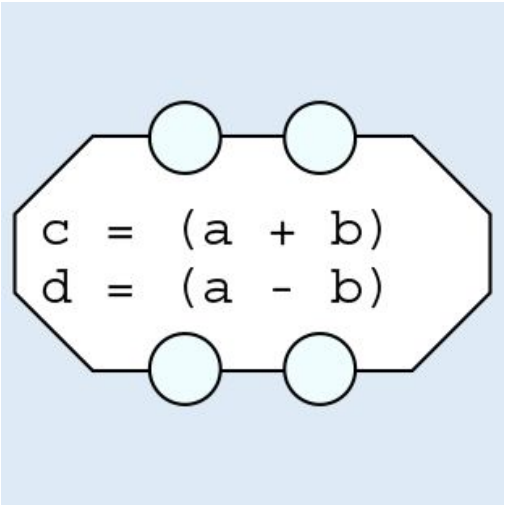

```
sdir.state @state_0 {
        // state body
}
```

SDIR: Access Nodes



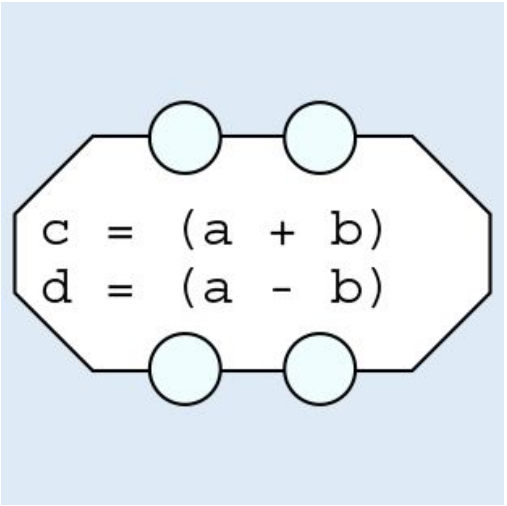
```
%A = sdir.alloc()
%C = sdir.alloc()

sdir.state @state_0 {
        %a = sdir.get_access %A
        %c = sdir.get_access %C
        sdir.copy %a -> %c
}
```

SDIR: Tasklets




```
sdir.tasklet{outputs=["c", "d"]} @add(%a, %b) -> (i32, i32) {
        %c = arith.addi %a, %b
        %d = arith.subi %a, %b
        sdir.return %c, %d
}
```

SDIR: Tasklets

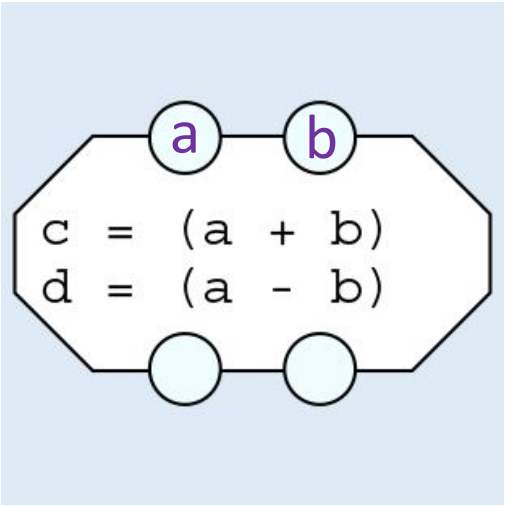



```
sdir.tasklet{outputs=["c", "d"]} @add(%a, %b) -> (i32, i32) {
        %c = arith.addi %a, %b
        %d = arith.subi %a, %b
        sdir.return %c, %d
}
```

Variable renaming

```
sdir.tasklet{outputs=["c", "d"]} @add(%a, %b) -> (i32, i32) {
        %r0 = arith.addi %a, %b
        %r1 = arith.subi %a, %b
        sdir.return %r0, %r1
}
```

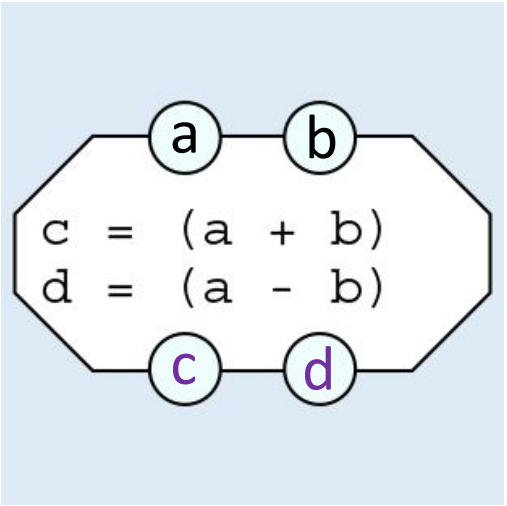
SDIR: Tasklets



Derive connector names

```
sdir.tasklet{outputs=["c", "d"]} @add(%a, %b) -> (i32, i32) {
        %r0 = arith.addi %a, %b
        %r1 = arith.subi %a, %b
        sdir.return %r0, %r1
}
```

SDIR: Tasklets

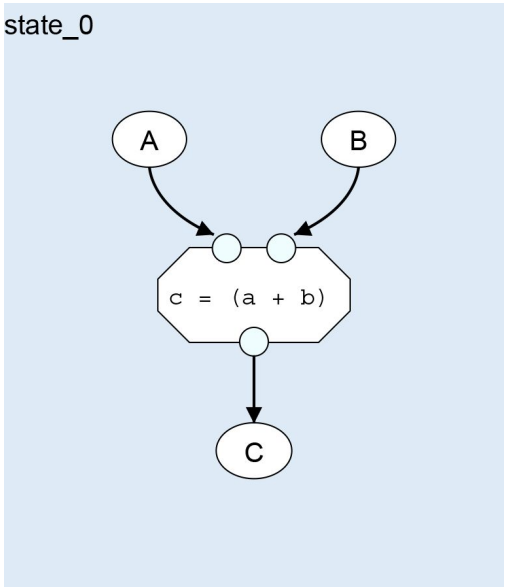


Derive connector names

```
sdir.tasklet{outputs=["c", "d"]} @add(%a, %b) -> (i32, i32) {
        %r0 = arith.addi %a, %b
        %r1 = arith.subi %a, %b
        sdir.return %r0, %r1
}
```

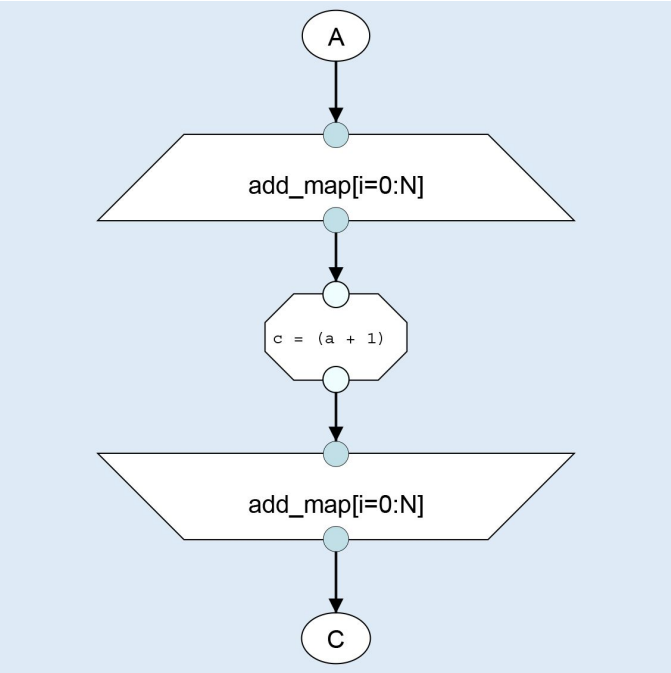
Connector names lost

Simple SDFG example



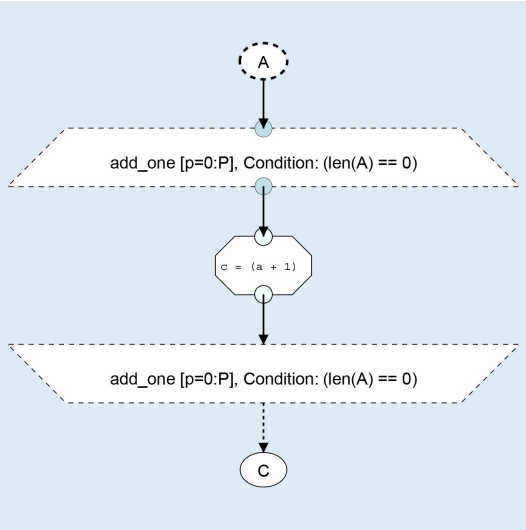

```
sdir.state @state_0 {
        sdir.tasklet @add(%a, %b) -> i32 {
                %c = arith.addi %a, %b
                sdir.return %c
        }

        %a = sdir.get_access %A
        %b = sdir.get_access %B
        %c = sdir.get_access %C

        %a_v = sdir.load %a[0]
        %b_v = sdir.load %b[0]
        %c_v = sdir.call @add(%a_v, %b_v)

        sdir.store %c_v, %c[0]
}
```

SDIR: Map Scopes



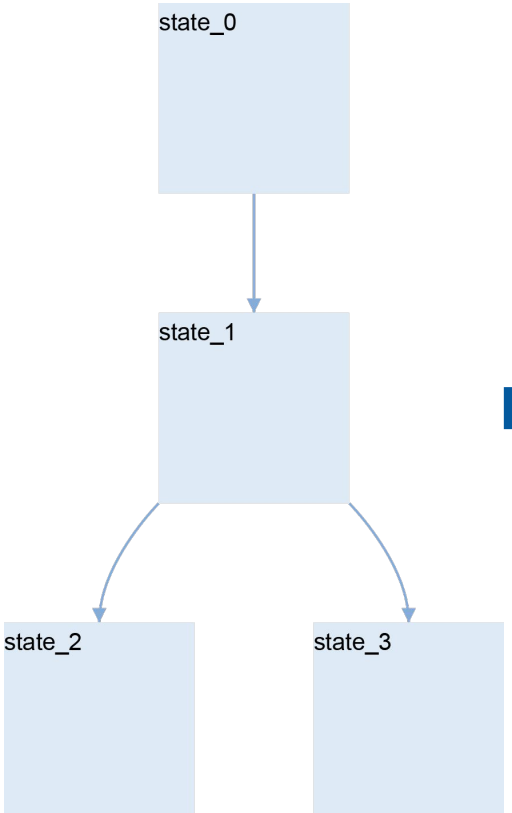

```
sdir.alloc_symbol("N")

sdir.map (%i) = (0) to (sym("N")) step (1) {
        %a = sdir.load %A[%i]
        %c = sdir.call @add_one(%a)
        sdir.store %c, %C[%i]
}
```

SDIR: Consume Scopes




```
%A = sdir.alloc_stream()
%S = sdir.get_access %A

sdir.consume{num_pes=5, condition=@empty} (%S) -> (pe: %p, elem: %e) {
        %c = sdir.call @add_one(%e)
        sdir.store{wcr="add"} %c, %C[0]
}
```

SDIR: SDFGs




```
sdir.sdfg{entry=@state_0} {
        sdir.edge{assign=["i: 1"]} @state_0 -> @state_1
        sdir.edge{condition="i > 1"} @state_1 -> @state_2
        sdir.edge{condition="i <= 1"} @state_1 -> @state_3
}
```

SDIR: Symbols

```
sdir.sdfg{entry=@state_0} {
        sdir.alloc_symbol("N")
}
```

SDIR: Symbols

```
sdir.sdfg{entry=@state_0} {
        sdir.alloc_symbol("N")
        …
        sdir.state @state_0 {
                %2 = sdir.sym("N") : index
        }
}
```

SDIR: Symbols

```
sdir.sdfg{entry=@state_0} {
        sdir.alloc_symbol("N")
        …
        sdir.state @state_0 {
                %2 = sdir.sym("2*N+5") : index
        }
}
```

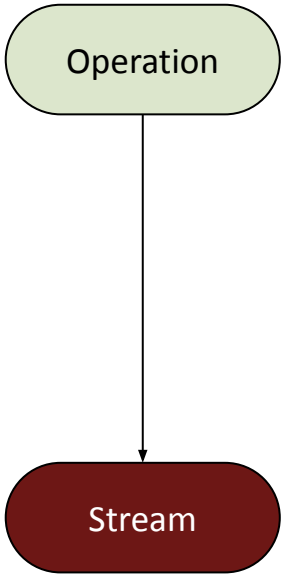
SDIR: Symbols

```
sdir.sdfg{entry=@state_0} {
        sdir.alloc_symbol("N")
        …
        sdir.state @state_0 {
                %2 = sdir.sym("N") : index
                sdir.store %2, %arg1[] : index -> !sdir.memlet<sym(“K”)xindex>
        }
}
```

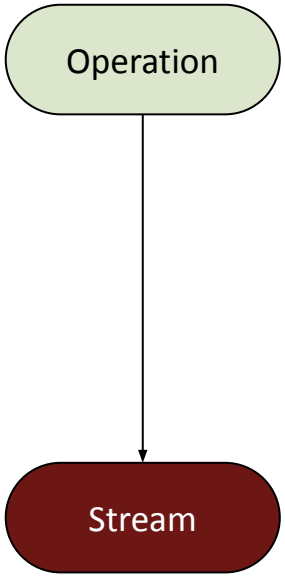
SDIR: Symbols

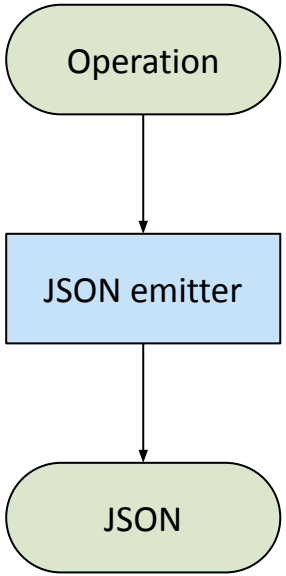
```
sdir.sdfg{entry=@state_0} {
        sdir.alloc_symbol("N")
        …
        sdir.state @state_0 {
                %2 = sdir.sym("N") : index
                sdir.store %2, %arg1[] : index -> !sdir.memlet<sym(“K”)xindex>
        }
        …
        sdir.edge{assign=[“N: 1”]} @state_1 -> @state_3
        sdir.edge{condition=“N <= 1”} @state_2 -> @state_3
}
```

JSON emitter



JSON emitter





SDIR -> SDFG Translator

```
%a = sdir.get_access %A
```

```
LogicalResult translateGetAccessToSDFG(GetAccessOp &op, JsonEmitter &jemit){
    jemit.startObject();
    jemit.printKVPair("type", "AccessNode");
    jemit.printKVPair("label", op.getName());

    jemit.startNamedObject("attributes");
    jemit.printKVPair("access", "ReadWrite");
    jemit.printKVPair("setzero", "false", false);
    jemit.printKVPair("data", op.getName());
    jemit.startNamedObject("in_connectors");
    jemit.endObject(); // in_connectors
    jemit.startNamedObject("out_connectors");
    jemit.endObject(); // out_connectors
    jemit.endObject(); // attributes

    jemit.printKVPair("id", op.ID(), false);
    jemit.printKVPair("scope_entry", "null", false);
    jemit.printKVPair("scope_exit", "null", false);
    jemit.endObject();

    return success();
}
```

SDIR -> SDFG Translator

```
{
        “type”: “AccessNode”,
        “label”: “A”,
        “attributes”: {
                “access”: “ReadWrite”,
                “setzero”: false,
                “data”: “A”,
                “in_connectors”: {
                },
                “out_connectors”: {
                }
        },
        “id”: 1,
        “scope_entry”: null,
        “scope_exit”: null
}
```

```
LogicalResult translateGetAccessToSDFG(GetAccessOp &op, JsonEmitter &jemit){
        jemit.startObject();
        jemit.printKVPair(“type”, “AccessNode”);
        jemit.printKVPair(“label”, op.getName());

        jemit.startNamedObject(“attributes”);
        jemit.printKVPair(“access”, “ReadWrite”);
        jemit.printKVPair(“setzero”, “false”, false);
        jemit.printKVPair(“data”, op.getName());
        jemit.startNamedObject(“in_connectors”);
        jemit.endObject(); // in_connectors
        jemit.startNamedObject(“out_connectors”);
        jemit.endObject(); // out_connectors
        jemit.endObject(); // attributes

        jemit.printKVPair(“id”, op.ID(), false);
        jemit.printKVPair(“scope_entry”, “null”, false);
        jemit.printKVPair(“scope_exit”, “null”, false);
        jemit.endObject();

        return success();
}
```

SDIR -> SDFG Translator

```
{
        “type”: “AccessNode”,
        “label”: “A”,
        “attributes”: {
                “access”: “ReadWrite”,
                “setzero”: false,
                “data”: “A”,
                “in_connectors”: {
                },
                “out_connectors”: {
                }
        },
        “id”: 1,
        “scope_entry”: null,
        “scope_exit”: null
}
```

```
LogicalResult translateGetAccessToSDFG(GetAccessOp &op, JsonEmitter &jemit){
        jemit.startObject();
                jemit.printKVPair(“type”, “AccessNode”);
                jemit.printKVPair(“label”, op.getName());

                jemit.startNamedObject(“attributes”);
                        jemit.printKVPair(“access”, “ReadWrite”);
                        jemit.printKVPair(“setzero”, “false”, false);
                        jemit.printKVPair(“data”, op.getName());
                        jemit.startNamedObject(“in_connectors”);
                        jemit.endObject(); // in_connectors
                        jemit.startNamedObject(“out_connectors”);
                        jemit.endObject(); // out_connectors
                jemit.endObject(); // attributes

                jemit.printKVPair(“id”, op.ID(), false);
                jemit.printKVPair(“scope_entry”, “null”, false);
                jemit.printKVPair(“scope_exit”, “null”, false);
        jemit.endObject();

        return success();
}
```

SDIR -> SDFG Translator

```
{
        “type”: “AccessNode”,
        “label”: “A”,
        “attributes”: {
                “access”: “ReadWrite”,
                “setzero”: false,
                “data”: “A”,
                “in_connectors”: {
                },
                “out_connectors”: {
                }
        },
        “id”: 1,
        “scope_entry”: null,
        “scope_exit”: null
}
```

```
LogicalResult translateGetAccessToSDFG(GetAccessOp &op, JsonEmitter &jemit){
        jemit.startObject();
                jemit.printKVPair(“type”, “AccessNode”);
                jemit.printKVPair(“label”, op.getName());

                jemit.startNamedObject(“attributes”);
                        jemit.printKVPair(“access”, “ReadWrite”);
                        jemit.printKVPair(“setzero”, “false”, false);
                        jemit.printKVPair(“data”, op.getName());
                        jemit.startNamedObject(“in_connectors”);
                        jemit.endObject(); // in_connectors
                        jemit.startNamedObject(“out_connectors”);
                        jemit.endObject(); // out_connectors
                jemit.endObject(); // attributes

                jemit.printKVPair(“id”, op.ID(), false);
                jemit.printKVPair(“scope_entry”, “null”, false);
                jemit.printKVPair(“scope_exit”, “null”, false);
        jemit.endObject();

        return success();
}
```

Recursive Translation

```
sdir.sdfg{entry=@state_0} {
        // sdfg body
}
```

```
[...]

for (Operation &oper : op.body().getOps())
        if (StateNode state = dyn_cast<StateNode>(oper))
                if (translateStateToSDFG(state, jemit).failed())
                        return failure();

[...]
```

Implicit top-level module operation

Lifting to Python

```
sdir.tasklet @add(%a, %b) -> f64 {
        %r = arith.addi %a, %b
        sdir.return %r
}
```

```
__out = (a + b)
```

```
sdir.tasklet @get_0() -> f64 {
        %r = arith.constant 0.0
        sdir.return %r
}
```

```
__out = dace.float(0.0)
```

Argument Allocation

```
sdir.sdfg @name(%arg0: i32, %arg1: i32, %arg2: i32) {
  …
  %c = sdir.call @add(%arg0, %arg0) : (i32, i32) -> i32
  …
}
```

Argument Allocation

```
sdir.sdfg @name(%arg0: i32, %arg1: i32, %arg2: i32) {
  …
  %c = sdir.call @add(%arg0, %arg0) : (i32, i32) -> i32
  …
}
```

```
sdir.sdfg @name(%arg0: i32, %arg1: i32, %arg2: i32) {
  %0 = sdir.alloc {name = "_arg2"}() : !sdir.array<i32>
  %1 = sdir.alloc {name = "_arg1"}() : !sdir.array<i32>
  %2 = sdir.alloc {name = "_arg0"}() : !sdir.array<i32>
  …
  %c = sdir.call @add(%arg0, %arg0) : (i32, i32) -> i32
  …
```

Argument Allocation

```
sdir.sdfg @name(%arg0: i32, %arg1: i32, %arg2: i32) {
  …
  %c = sdir.call @add(%arg0, %arg0) : (i32, i32) -> i32
  …
}
```

```
sdir.sdfg @name(%arg0: i32, %arg1: i32, %arg2: i32) {
  %0 = sdir.alloc {name = "_arg2"}() : !sdir.array<i32>
  %1 = sdir.alloc {name = "_arg1"}() : !sdir.array<i32>
  %2 = sdir.alloc {name = "_arg0"}() : !sdir.array<i32>
  …
  %a = sdir.get_access %2 : !sdir.array<i32> -> !sdir.memlet<i32>
  %b = sdir.load %a[] : !sdir.memlet<i32> -> i32
  %c = sdir.call @add(%b, %b) : (i32, i32) -> i32
  …
```

Dead loads

```
sdir.state @state_1 {
  sdir.tasklet @task_3() -> index {
    %c0 = arith.constant 0 : index
    sdir.return %c0 : index
  }
  %6 = sdir.call @task_3() : () -> index
  %7 = sdir.get_access %5 : !sdir.array<index> -> !sdir.memlet<index>
  sdir.store %6, %7[] : index -> !sdir.memlet<index>
  %8 = sdir.load %7[] : !sdir.memlet<index> -> index
}
```

Dead loads

```
sdir.state @state_1 {
  sdir.tasklet @task_3() -> index {
    %c0 = arith.constant 0 : index
    sdir.return %c0 : index
  }
  %6 = sdir.call @task_3() : () -> index
  %7 = sdir.get_access %5 : !sdir.array<index> -> !sdir.memlet<index>
  sdir.store %6, %7[] : index -> !sdir.memlet<index>
  %8 = sdir.load %7[] : !sdir.memlet<index> -> index
}
```

Dead loads

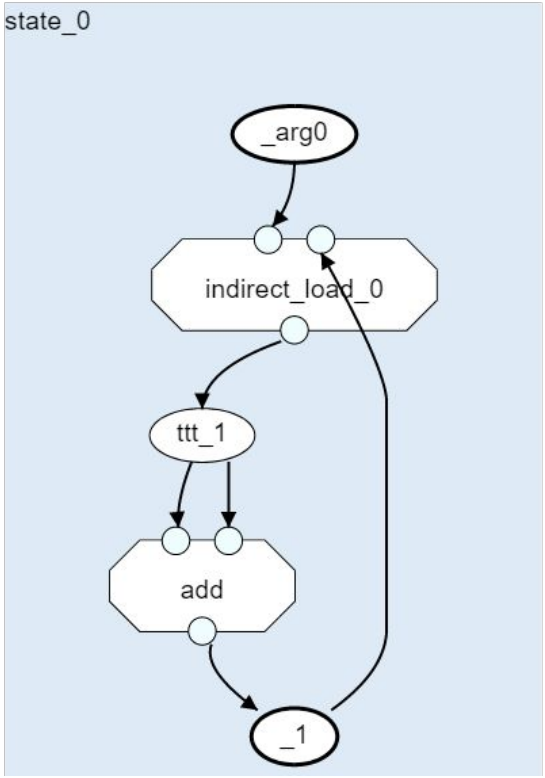
```
sdir.state @state_1 {
  sdir.tasklet @task_3() -> index {
    %c0 = arith.constant 0 : index
    sdir.return %c0 : index
  }
  %6 = sdir.call @task_3() : () -> index
  %7 = sdir.get_access %5 : !sdir.array<index> -> !sdir.memlet<index>
  sdir.store %6, %7[] : index -> !sdir.memlet<index>
}
```

Access Nodes

```
%a = sdir.get_access %A : !sdir.array<5xi32> -> !sdir.memlet<5xi32>
%b = sdir.load %a[%ind] : !sdir.memlet<5xi32> -> i32
%c = sdir.call @add(%b, %b) : (i32, i32) -> i32
sdir.store %c, %a[0] : i32 -> !sdir.memlet<5xi32>
```

Access Nodes

```
%a = sdir.get_access %A : !sdir.array<5xi32> -> !sdir.memlet<5xi32>
%b = sdir.load %a[%ind] : !sdir.memlet<5xi32> -> i32
%c = sdir.call @add(%b, %b) : (i32, i32) -> i32
sdir.store %c, %a[0] : i32 -> !sdir.memlet<5xi32>
```

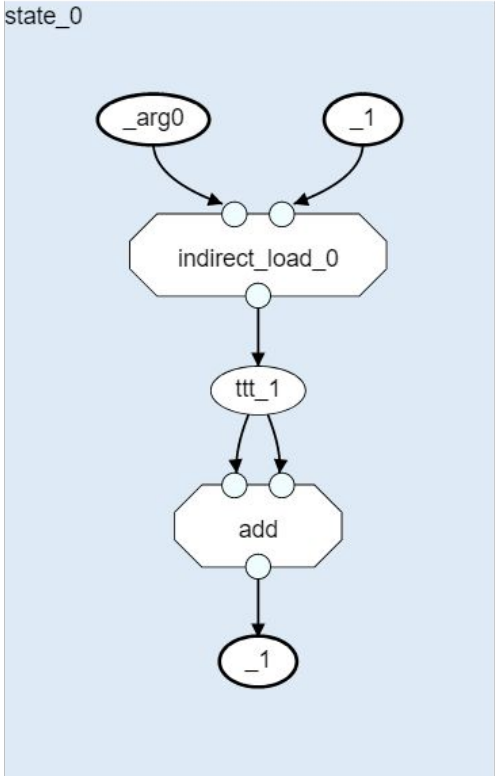


Access Nodes

```
%a = sdir.get_access %A : !sdir.array<5xi32> -> !sdir.memlet<5xi32>
%a2 = sdir.get_access %A : !sdir.array<5xi32> -> !sdir.memlet<5xi32>
%b = sdir.load %a[%ind] : !sdir.memlet<5xi32> -> i32
%c = sdir.call @add(%b, %b) : (i32, i32) -> i32
sdir.store %c, %a2[0] : i32 -> !sdir.memlet<5xi32>
```

Access Nodes

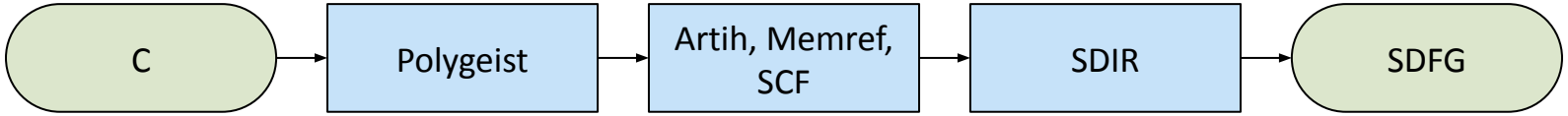
```
%a = sdir.get_access %A : !sdir.array<5xi32> -> !sdir.memlet<5xi32>
%a2 = sdir.get_access %A : !sdir.array<5xi32> -> !sdir.memlet<5xi32>
%b = sdir.load %a[%ind] : !sdir.memlet<5xi32> -> i32
%c = sdir.call @add(%b, %b) : (i32, i32) -> i32
sdir.store %c, %a2[0] : i32 -> !sdir.memlet<5xi32>
```



Converter

Converter


Arith
Memref
SCF
SDIR


C
Polygeist
Artih, Memref, SCF
SDIR
SDFG

Converter: Funcs

```
func private @binops(%arg0: i32) {
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Funcs

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  sdir.state @init_0{}
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  sdir.state @init_0{}
  sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
      %5 = arith.addi %arg2, %arg2 : i32
      sdir.return %5 : i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
     %5 = arith.addi %arg2, %arg2 : i32
     sdir.return %5 : i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %c0 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %4 : i32
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c0 = arith.addi %arg0, %arg0 : i32
  %c1 = arith.muli %arg0, %4 : i32
  sdir.edge @init_0 -> @state_1
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  sdir.state @init_0{}
  sdir.state @state_1 {
    sdir.tasklet @task_3(%arg1: i32, %arg2: i32) -> i32 {
       %5 = arith.addi %arg2, %arg2 : i32
       sdir.return %5 : i32
    }
    %2 = sdir.call @task_3(%arg0, %arg0) : (i32, i32) -> i32
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    sdir.store %2, %3[] : i32 -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }
  %c1 = arith.muli %arg0, %4 : i32
  sdir.edge @init_0 -> @state_1
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
  }
  %c1 = arith.muli %arg0, %4 : i32
  sdir.edge @init_0 -> @state_1
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
  }
  %c1 = arith.muli %arg0, %4 : i32
  sdir.edge @init_0 -> @state_1
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
  }
  %c1 = arith.muli %arg0, %4 : i32
  sdir.edge @init_0 -> @state_1
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
  }
  %c1 = arith.muli %arg0, %4 : i32
  sdir.edge @init_0 -> @state_1
  return
}
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
    %2 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %3 = sdir.load %2[] : !sdir.memlet<i32> -> i32
    %4 = sdir.call @task_6(%arg0, %3) : (i32, i32) -> i32
    …
  }
  %c1 = arith.muli %arg0, %4 : i32
 …
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
    %2 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %3 = sdir.load %2[] : !sdir.memlet<i32> -> i32
    %4 = sdir.call @task_6(%arg0, %3) : (i32, i32) -> i32
    …
  }
  %c1 = arith.muli %arg0, %4 : i32
 …
```

Converter: Arith Operations

```
sdir.sdfg {entry = @init_0} @binops(%arg0: i32) {
  %1 = sdir.alloc_transient {name = "_tmp_2"}() : !sdir.array<i32>
  …
  sdir.state @state_1 {
    …
    %3 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  }

  sdir.state @state_4 {
    …
    %2 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
    %3 = sdir.load %2[] : !sdir.memlet<i32> -> i32
    %4 = sdir.call @task_6(%arg0, %3) : (i32, i32) -> i32
    …
  }
  sdir.edge @state_1 -> @state_4
 …
```

Converter: Memref Operations

```
%8 = memref.load %arg1[%c0, %arg0] : memref<?x900xi32>
```

```
sdir.state @state_6 {
  %3 = sdir.get_access %2 : !sdir.array<index> -> !sdir.memlet<index>
  %4 = sdir.load %3[] : !sdir.memlet<index> -> index
  %5 = sdir.load %arg1[%4, %arg0] : !sdir.memlet<sym("s_0")x900xi32> -> i32
  %6 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
  sdir.store %5, %6[] : i32 -> !sdir.memlet<i32>
  %7 = sdir.load %6[] : !sdir.memlet<i32> -> i32
}
```

Converter: Memref Operations

```
%8 = memref.load %arg1[%c0, %arg0] : memref<?x900xi32>
```

```
sdir.state @state_6 {
  %3 = sdir.get_access %2 : !sdir.array<index> -> !sdir.memlet<index>
  %4 = sdir.load %3[] : !sdir.memlet<index> -> index
  %5 = sdir.load %arg1[%4, %arg0] : !sdir.memlet<sym("s_0")x900xi32> -> i32
  %6 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
  sdir.store %5, %6[] : i32 -> !sdir.memlet<i32>
  %7 = sdir.load %6[] : !sdir.memlet<i32> -> i32
}
```

Converter: Memref Operations

```
%8 = memref.load %arg1[%c0, %arg0] : memref<?x900xi32>
```

```
sdir.state @state_6 {
  %3 = sdir.get_access %2 : !sdir.array<index> -> !sdir.memlet<index>
  %4 = sdir.load %3[] : !sdir.memlet<index> -> index
  %5 = sdir.load %arg1[%4, %arg0] : !sdir.memlet<sym("s_0")x900xi32> -> i32
  %6 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
  sdir.store %5, %6[] : i32 -> !sdir.memlet<i32>
  %7 = sdir.load %6[] : !sdir.memlet<i32> -> i32
}
```

Converter: Memref Operations

```
%8 = memref.load %arg1[%c0, %arg0] : memref<?x900xi32>
```

```
sdir.state @state_6 {
  %3 = sdir.get_access %2 : !sdir.array<index> -> !sdir.memlet<index>
  %4 = sdir.load %3[] : !sdir.memlet<index> -> index
  %5 = sdir.load %arg1[%4, %arg0] : !sdir.memlet<sym("s_0")x900xi32> -> i32
  %6 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
  sdir.store %5, %6[] : i32 -> !sdir.memlet<i32>
  %7 = sdir.load %6[] : !sdir.memlet<i32> -> i32
}
```

Converter: Memref Operations

```
%8 = memref.load %arg1[%c0, %arg0] : memref<?x900xi32>
```

```
sdir.state @state_6 {
  %3 = sdir.get_access %2 : !sdir.array<index> -> !sdir.memlet<index>
  %4 = sdir.load %3[] : !sdir.memlet<index> -> index
  %5 = sdir.load %arg1[%4, %arg0] : !sdir.memlet<sym("s_0")x900xi32> -> i32
  %6 = sdir.get_access %1 : !sdir.array<i32> -> !sdir.memlet<i32>
  sdir.store %5, %6[] : i32 -> !sdir.memlet<i32>
  %7 = sdir.load %6[] : !sdir.memlet<i32> -> i32
}
```

Converter: Memref Operations

```
memref.store %c2, %arg1[%arg0, %arg0] : memref<?x900xi32>
```

```
sdir.state @state_10 {
  %3 = sdir.get_access %0 : !sdir.array<i32> -> !sdir.memlet<i32>
  %4 = sdir.load %3[] : !sdir.memlet<i32> -> i32
  sdir.store %4, %arg1[%arg0, %arg0] : i32 -> !sdir.memlet<sym("s_0")x900xi32>
}
```

Converter: SCF For Loops

```
scf.for %arg11 = %c0 to %c1 step %c2 {
  // Loop Body
}
```

Converter: SCF For Loops

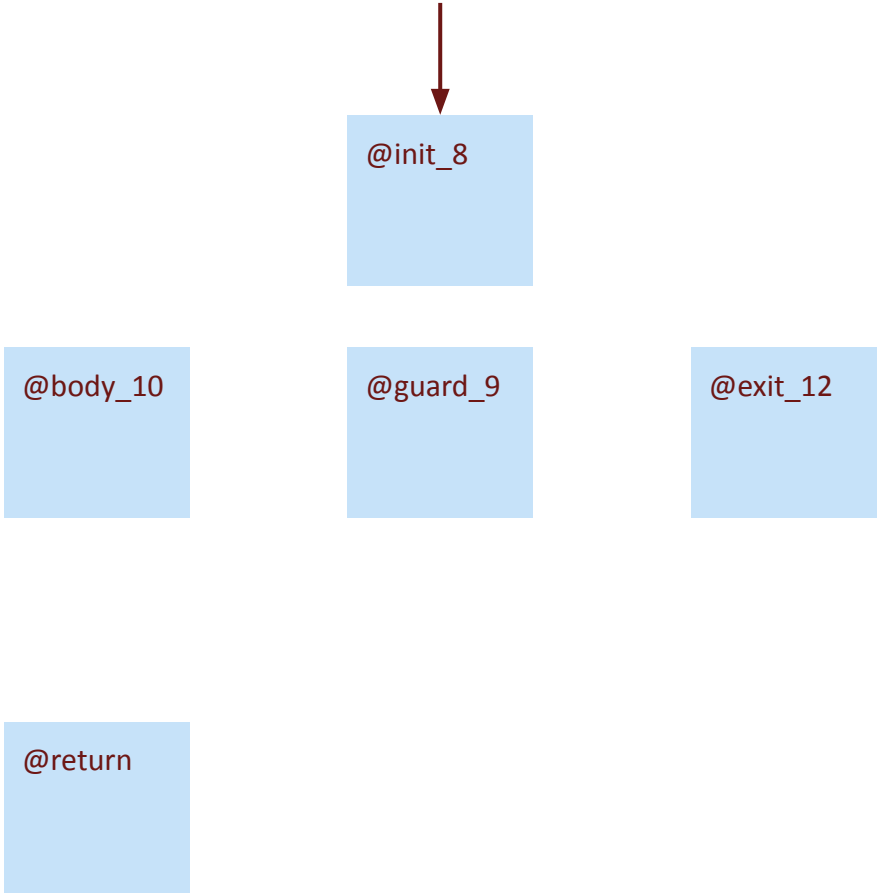
```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```

Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```

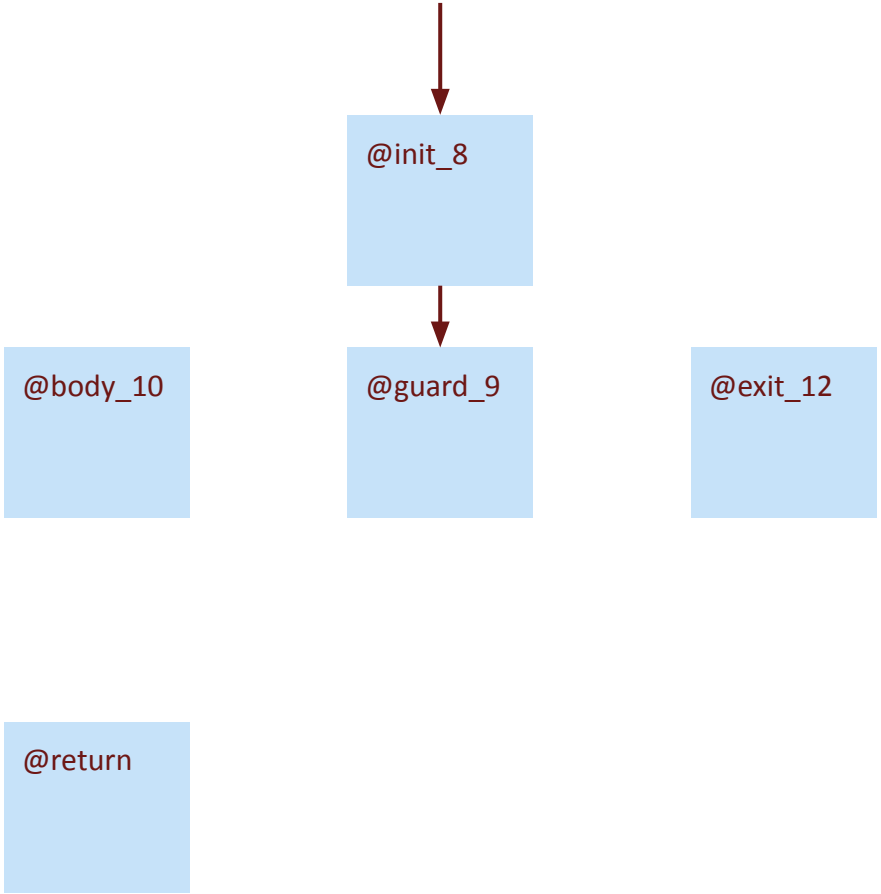
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



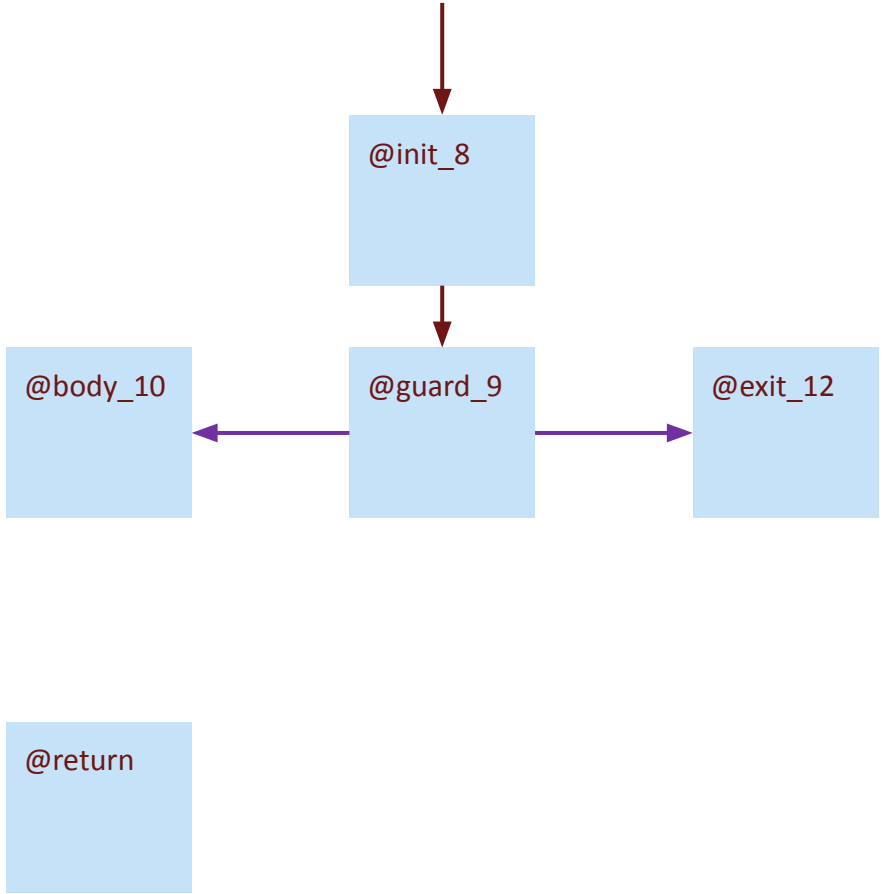
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



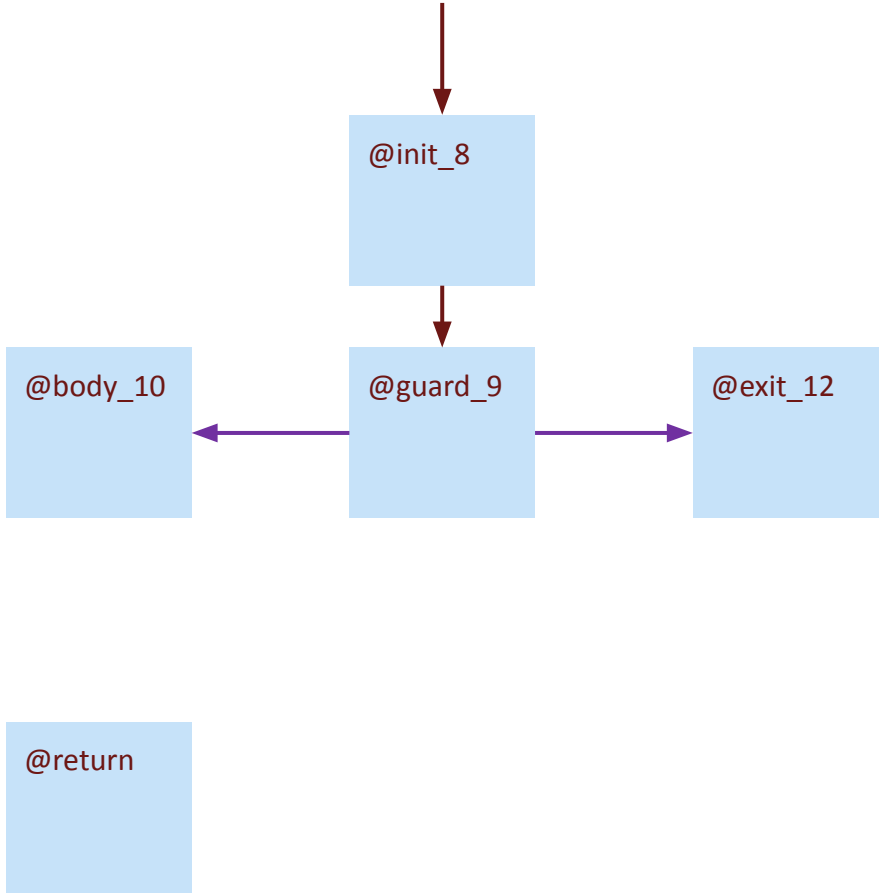
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



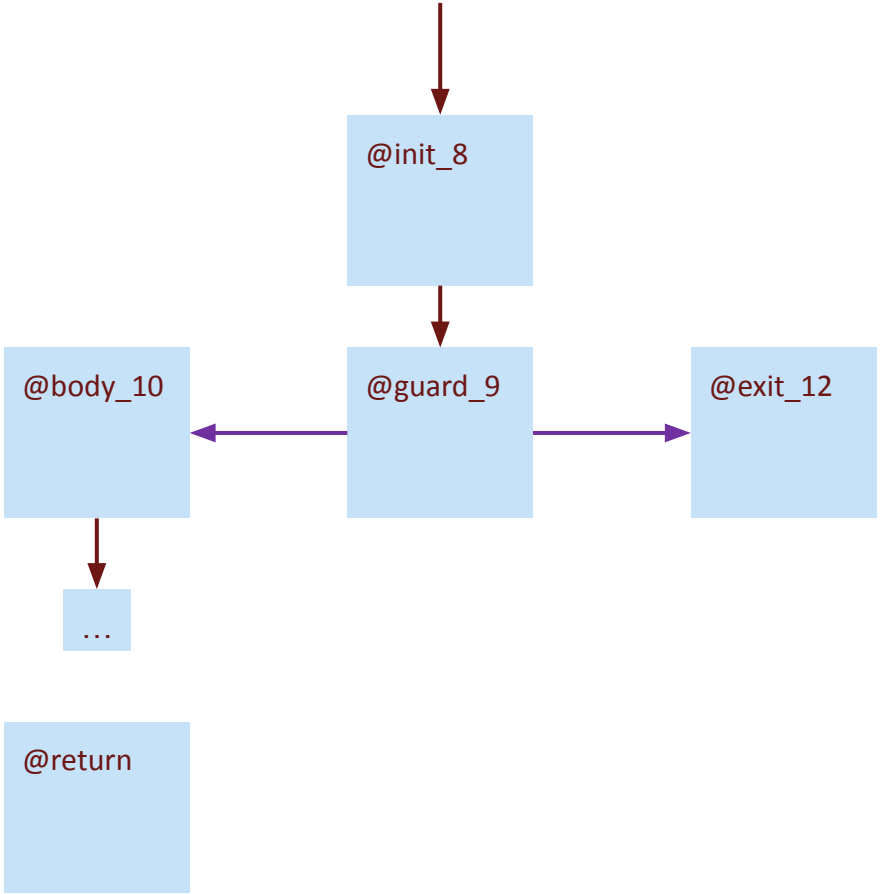
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



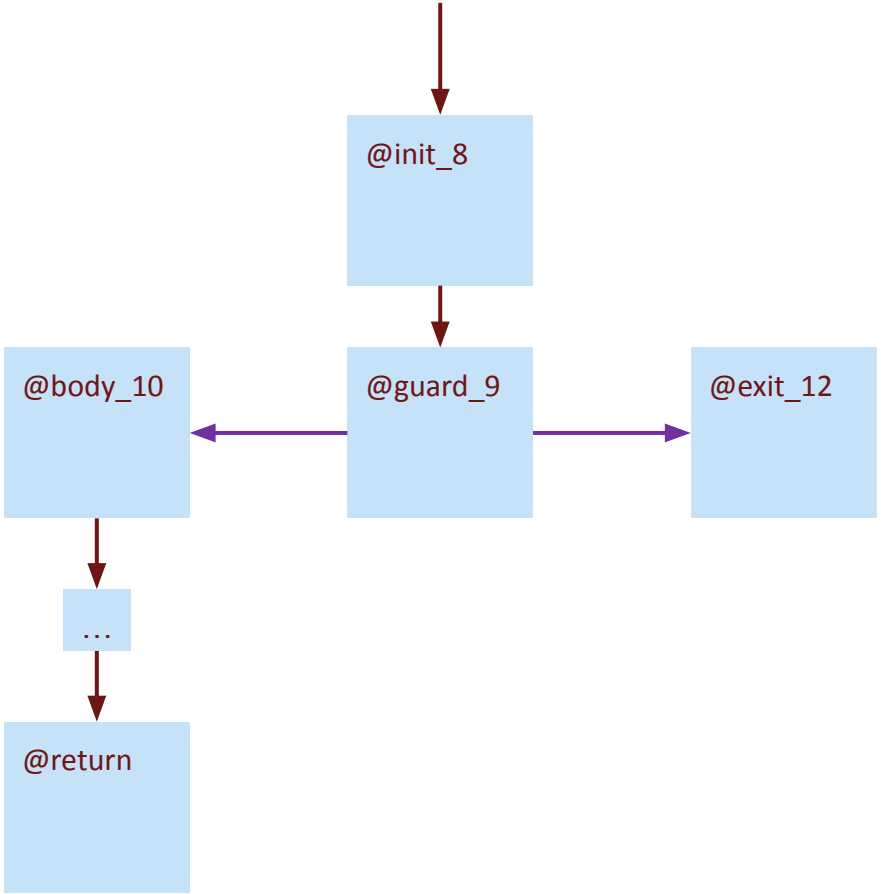
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



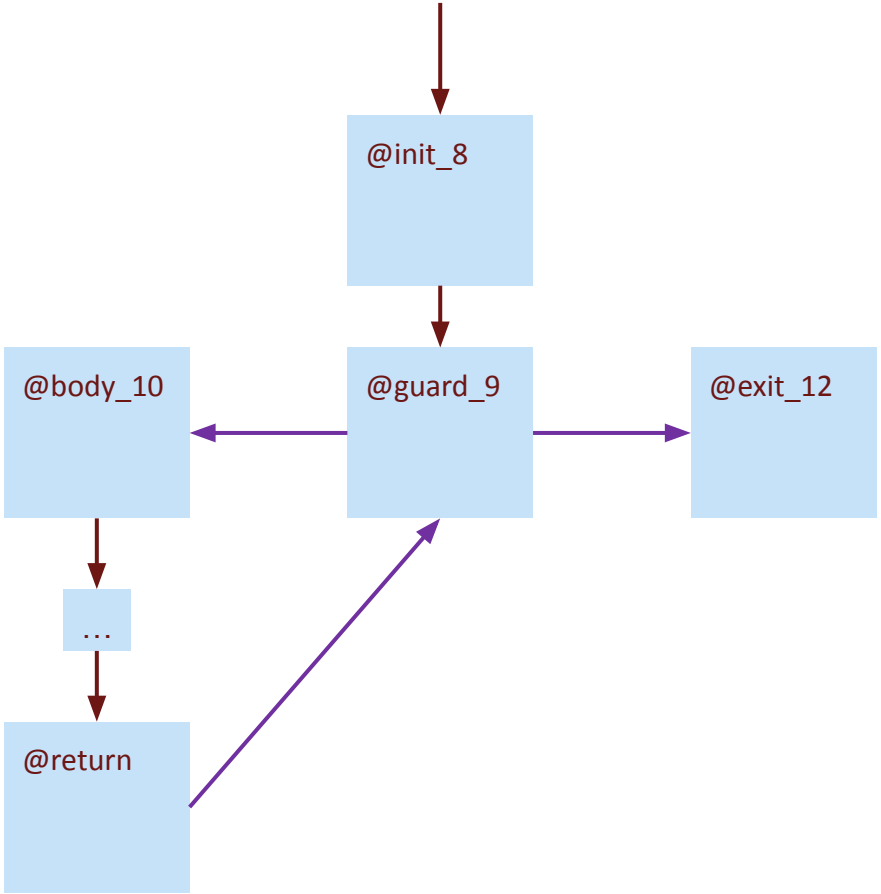
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



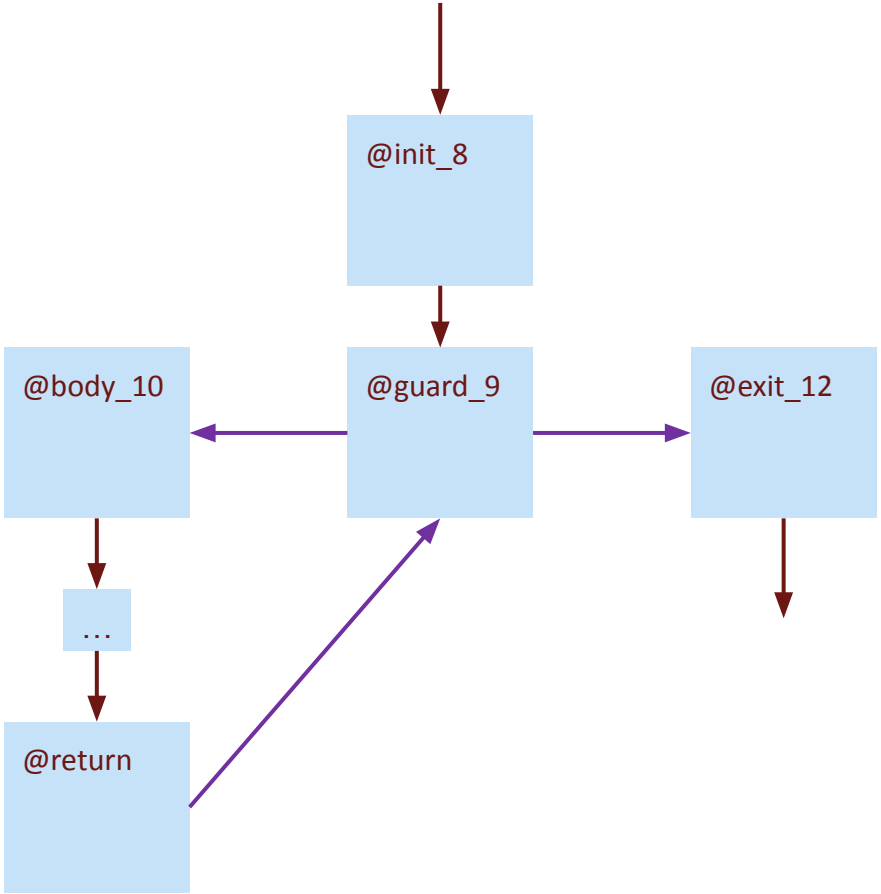
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



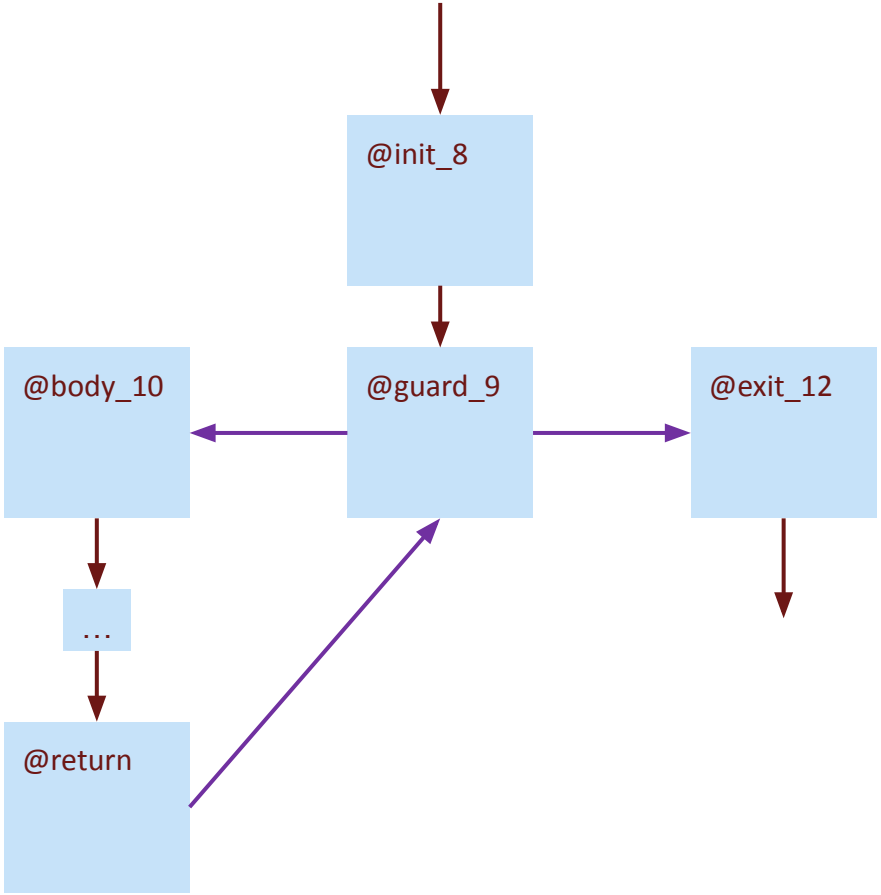
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



Converter: SCF For Loops

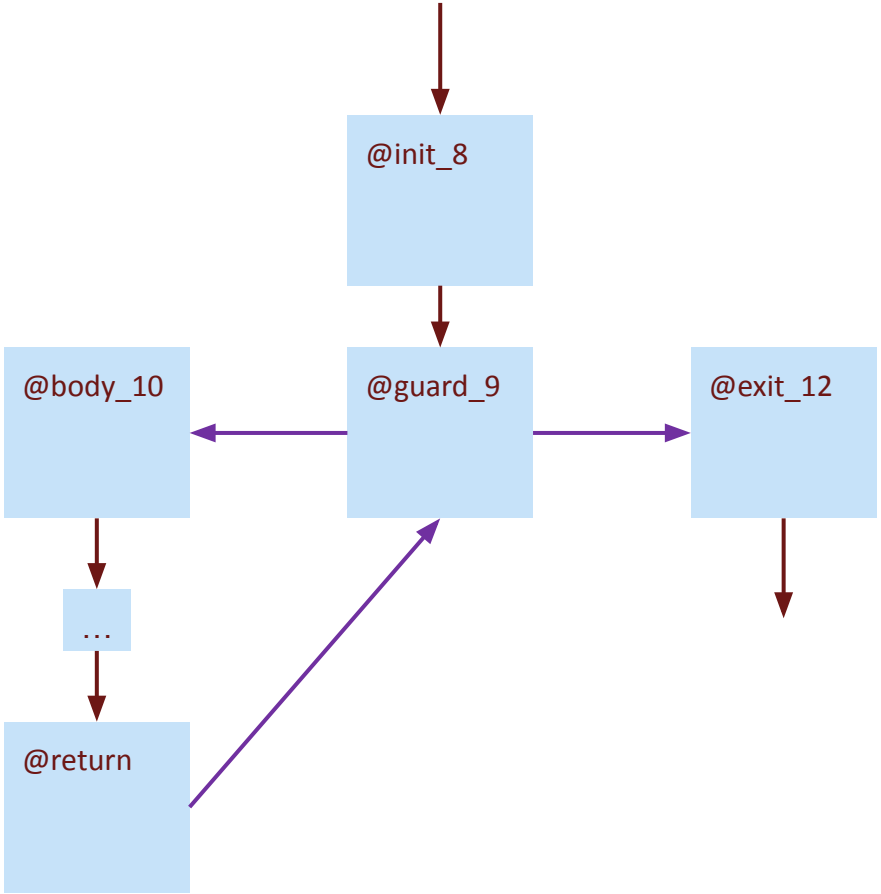
```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body
last.op %x, %y : t1

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



Converter: SCF For Loops

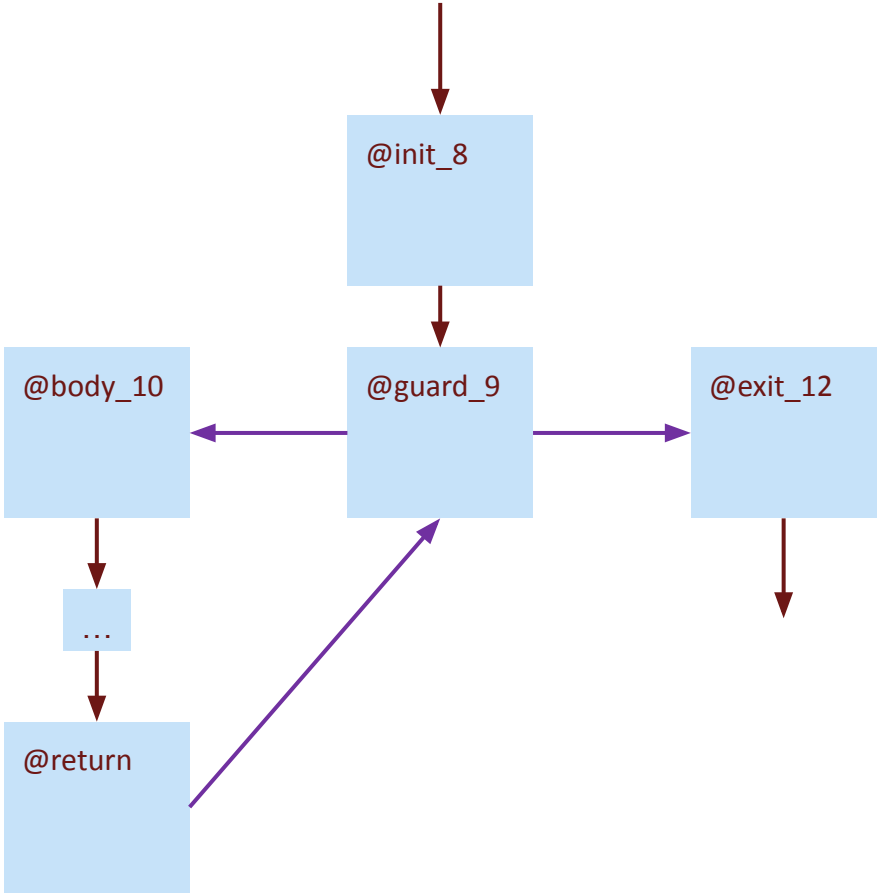
```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body
last.op %x, %y : t1

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```

Mark to link



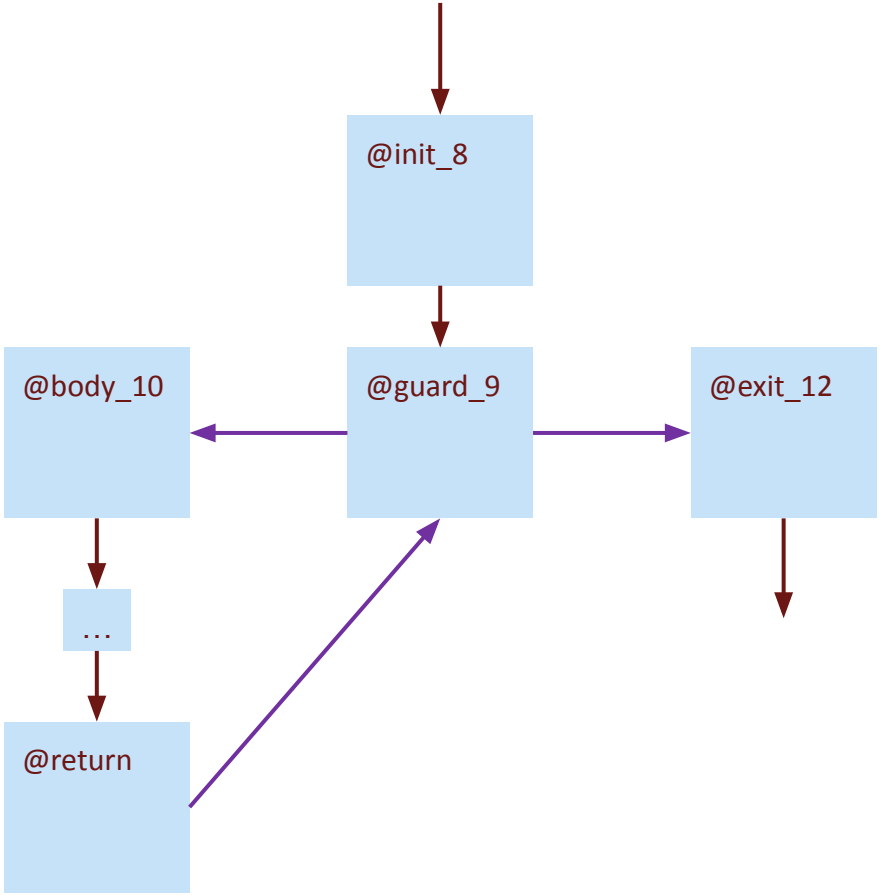
Converter: SCF For Loops

```
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body
last.op {linkToNext=true} %x, %y : t1

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



Converter: SCF For Loops

```
sdir.alloc_symbol("idx")

sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body
last.op {linkToNext=true} %x, %y : t1

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



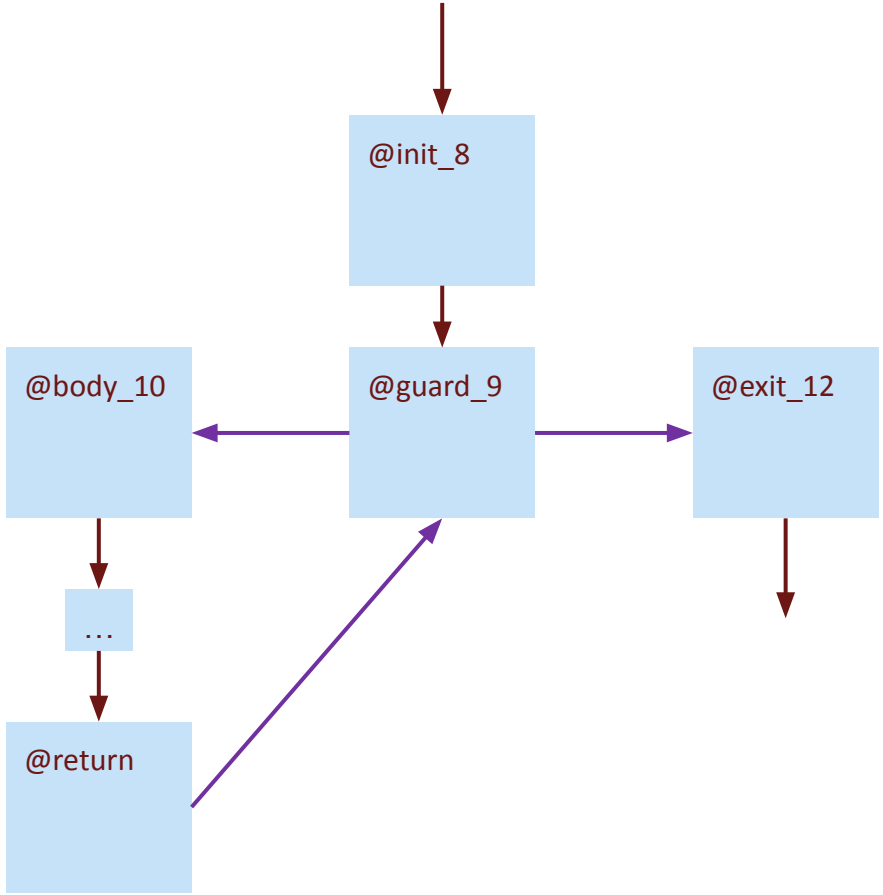
Converter: SCF For Loops

```
sdir.alloc_symbol("idx")

sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

// Loop Body
last.op {linkToNext=true} %x, %y : t1

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```

Inits “idx”

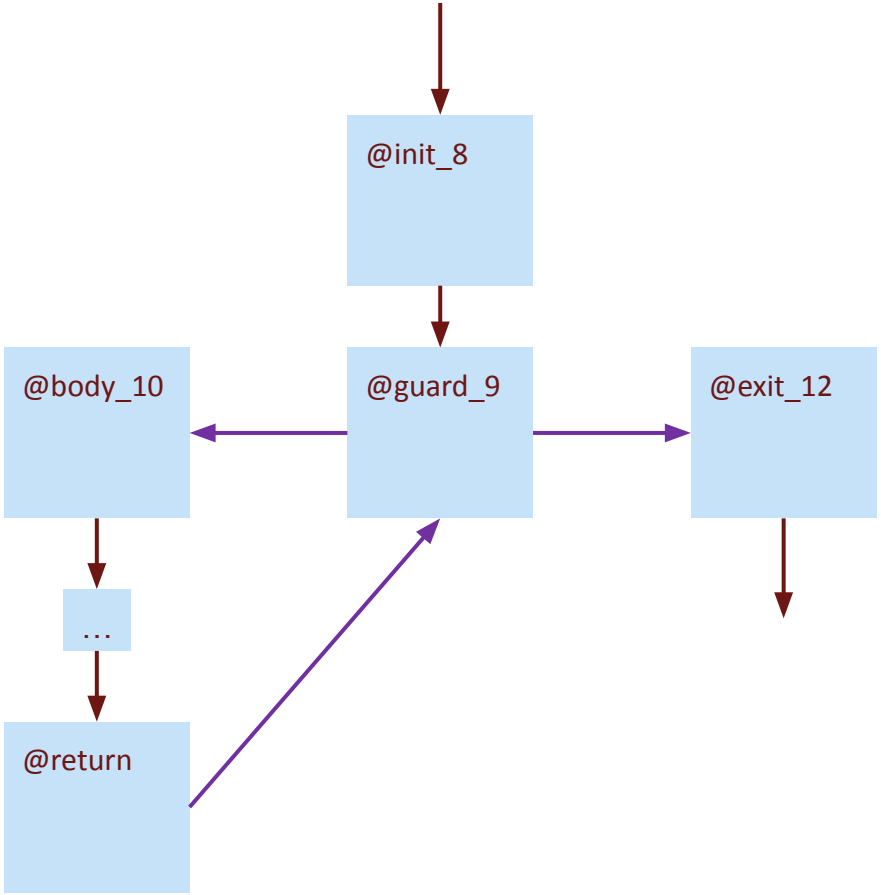
Checks “idx”

Increases“idx”



Converter: SCF For Loops

```
sdir.alloc_symbol("idx")

sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}

%2 = sdir.sym("idx") : index

// Loop Body
last.op {linkToNext=true} %x, %y : t1

sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
```



Outline

- Motivation
- SDIR
- **Evaluation**
- Summary

Methodology

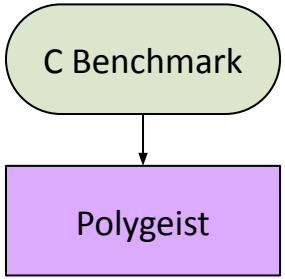
- PolyBench's 2mm, adi, gemver, heat-3d, trmm

Methodology

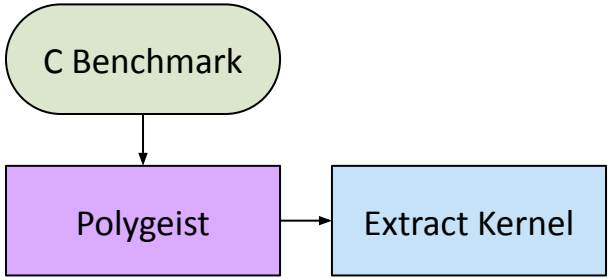
- PolyBench's 2mm, adi, gemver, heat-3d, trmm
- CSCS Ault server
- Build compilation pipeline for SDIR
- Compare with clang, gcc, polly, pluto
- Single- & Multi-Threaded
- Run 100 times on the large dataset
- 10 Runs warm-up

Architecture	x86_64
CPUs	72
Cores per socket	18
Base frequency	3.00 GHz
L1I cache size	32kB
L1D cache size	32kB
LLVM/MLIR version	14.0.0git
Clang version	10.0.1
DaCe version	0.11.4

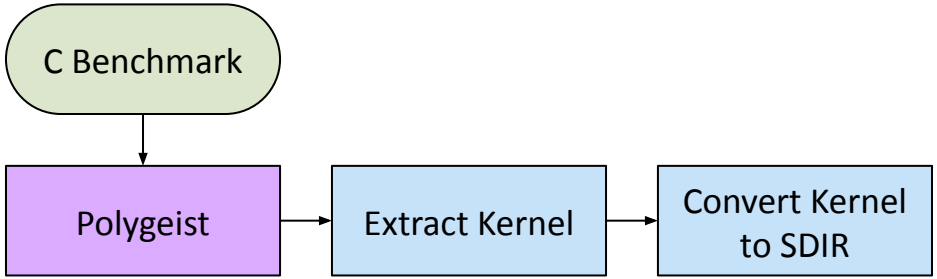
Compilation Pipeline



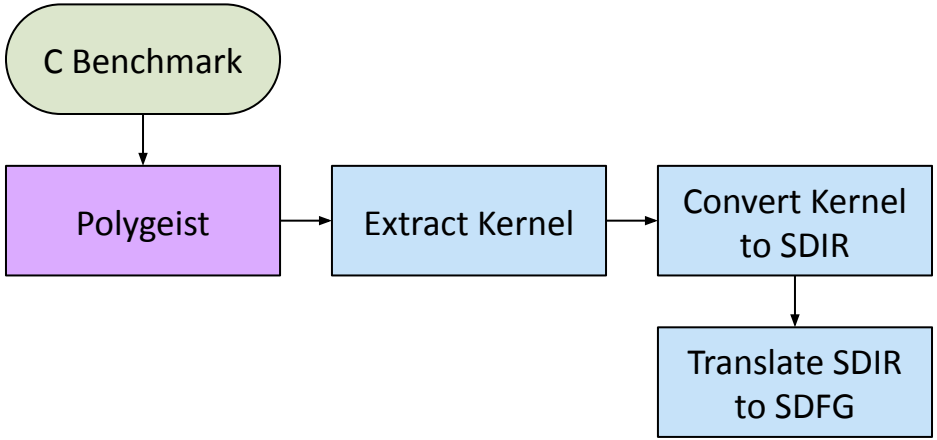
Compilation Pipeline



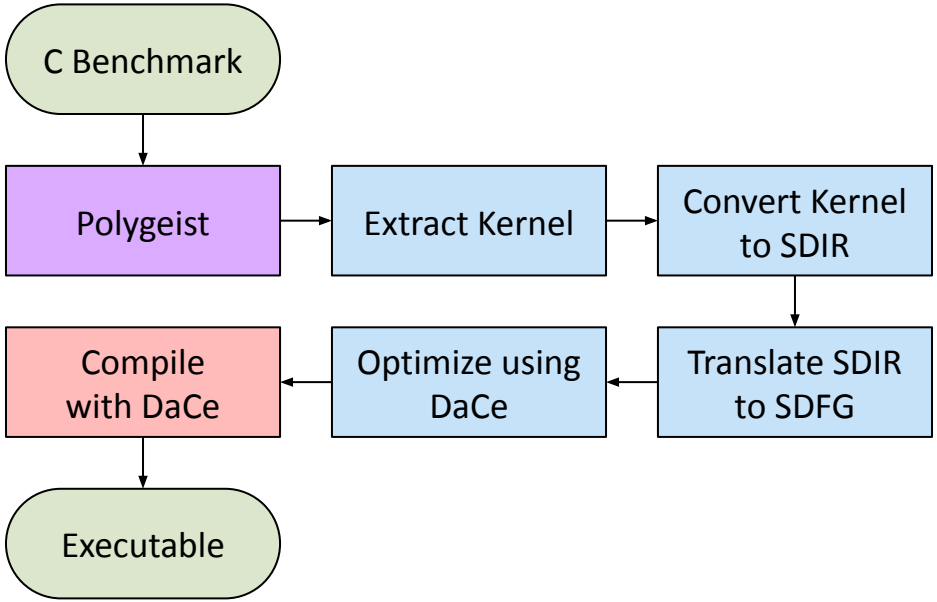
Compilation Pipeline



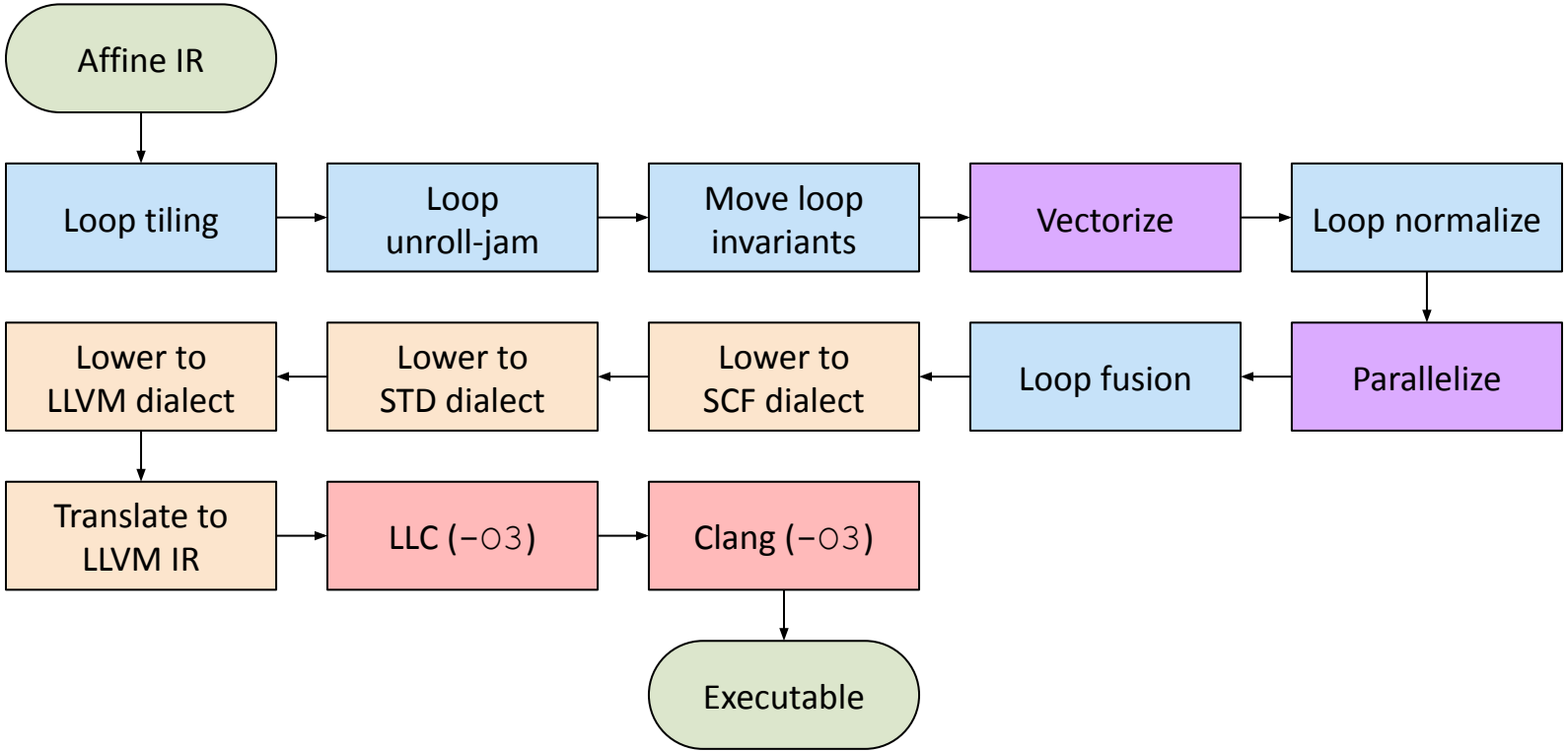
Compilation Pipeline



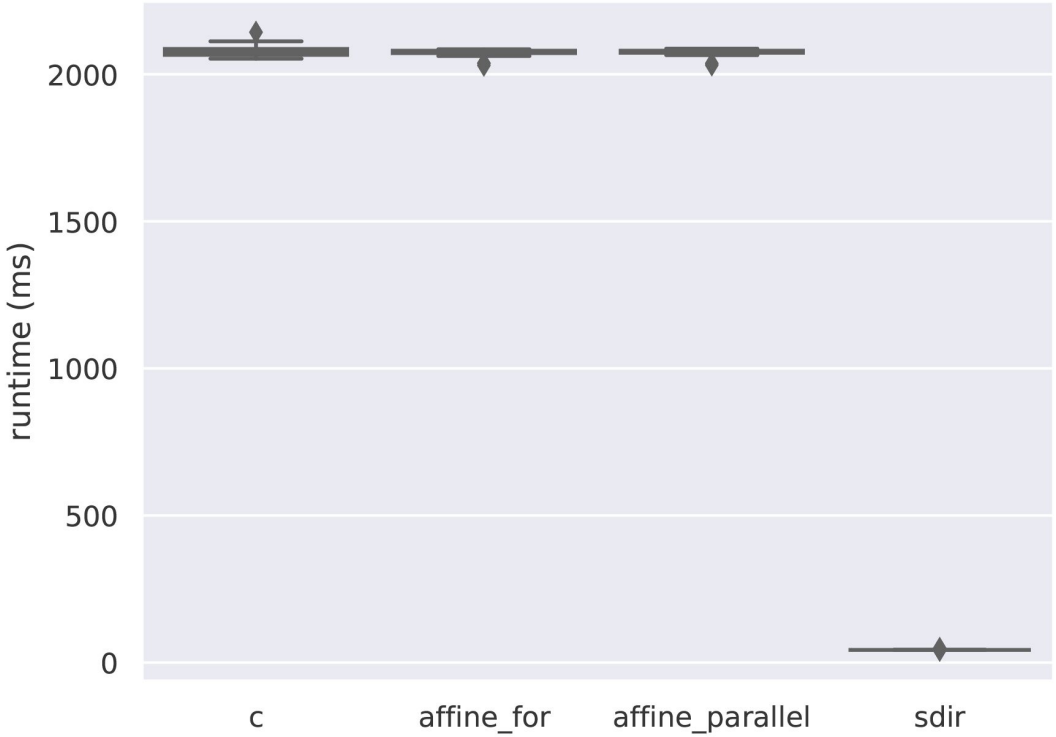
Compilation Pipeline



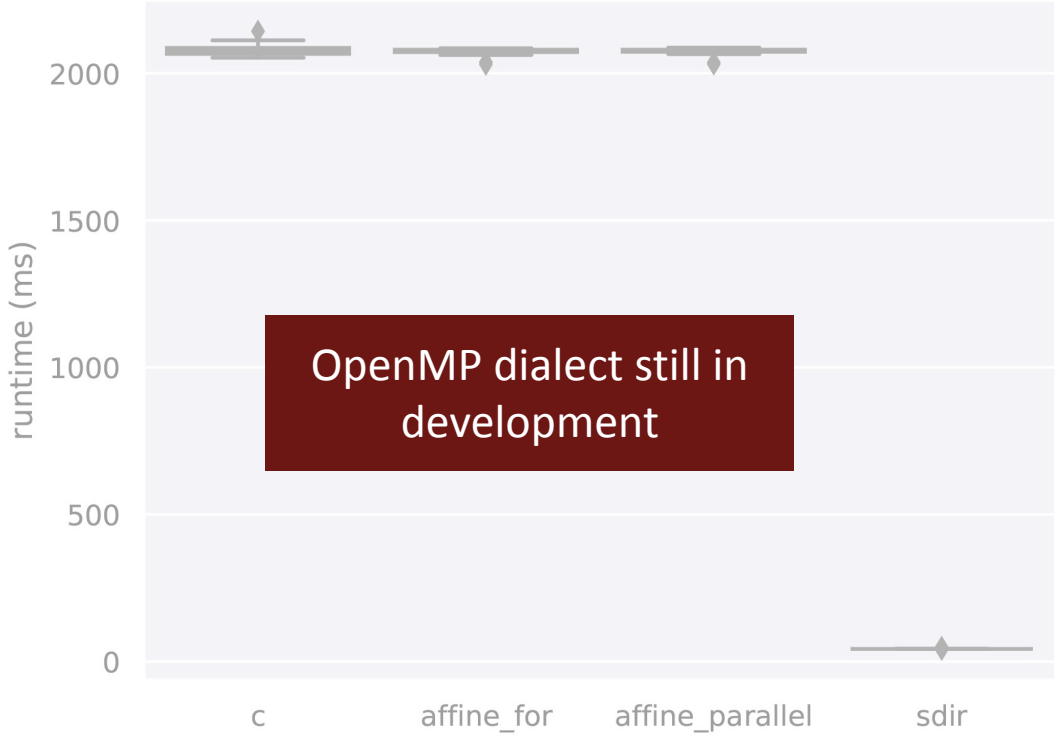
Affine pipeline



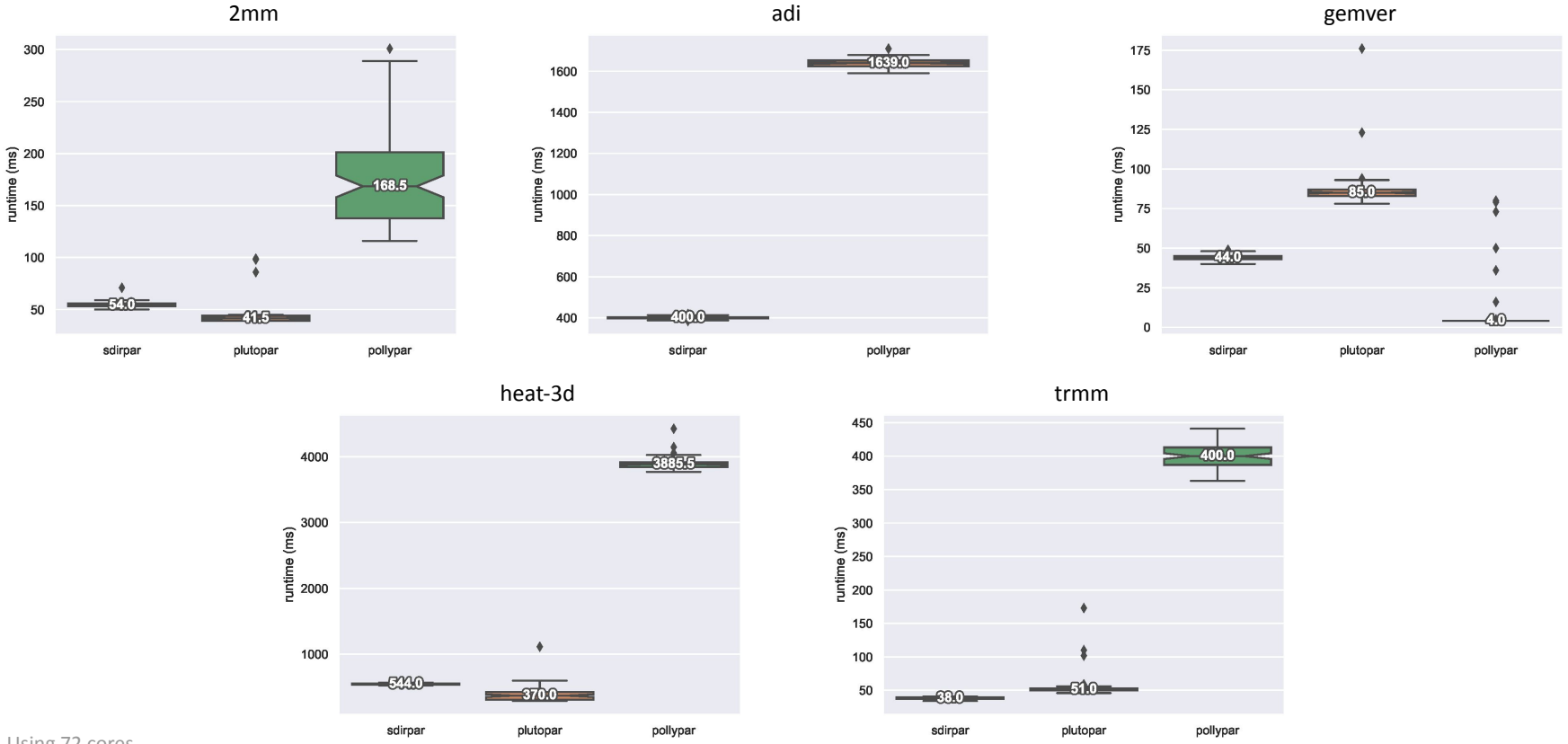
2MM Affine vs. SDIR



2MM Affine vs. SDIR

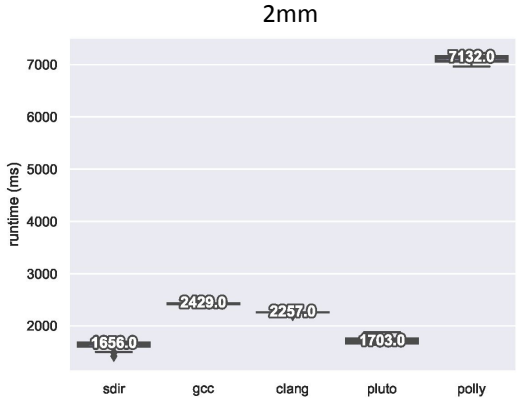


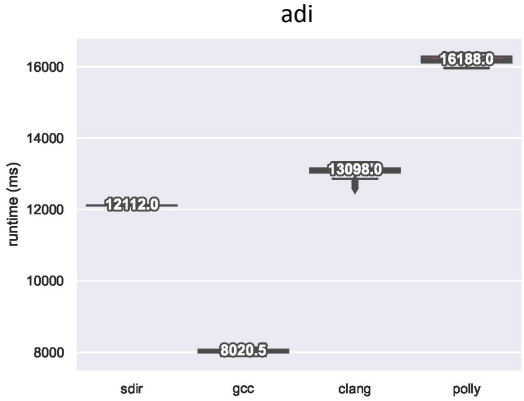
Parallel Benchmarks

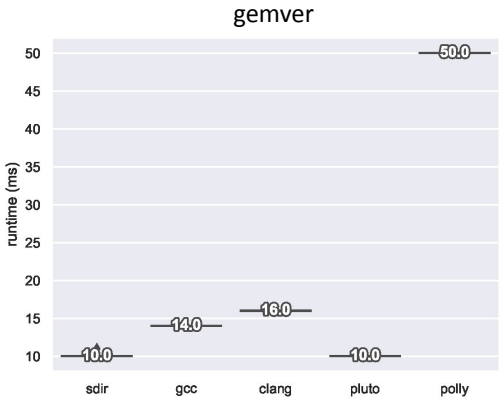


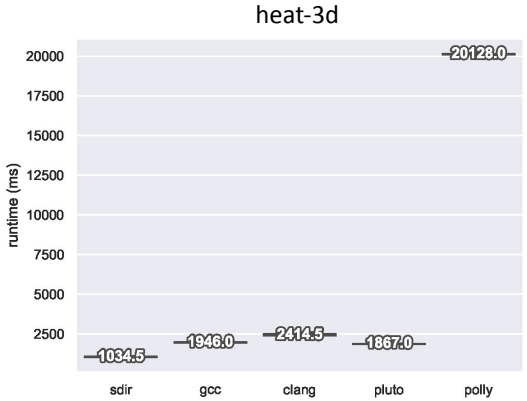
Using 72 cores

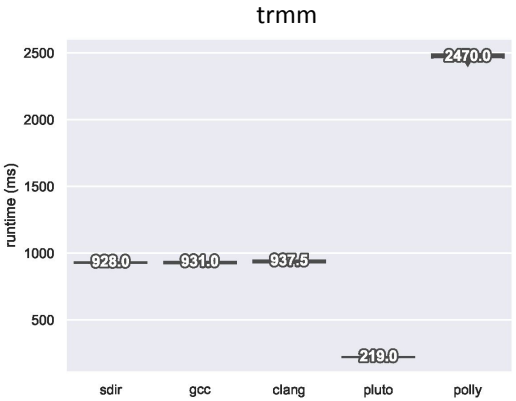
Sequential Benchmarks







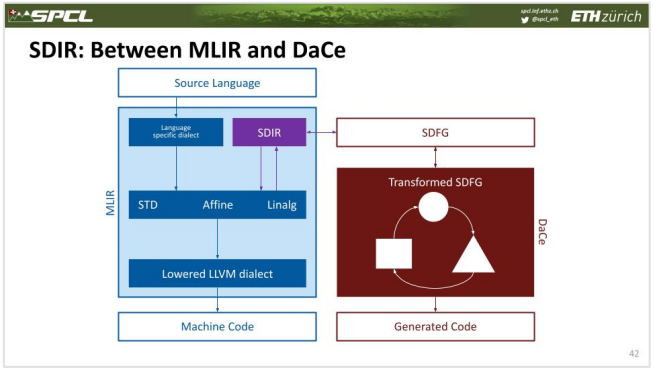


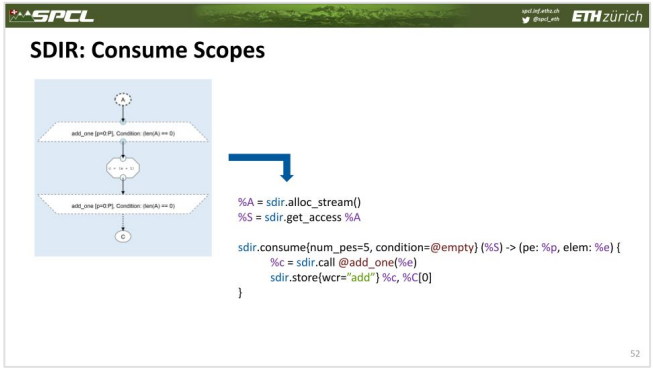


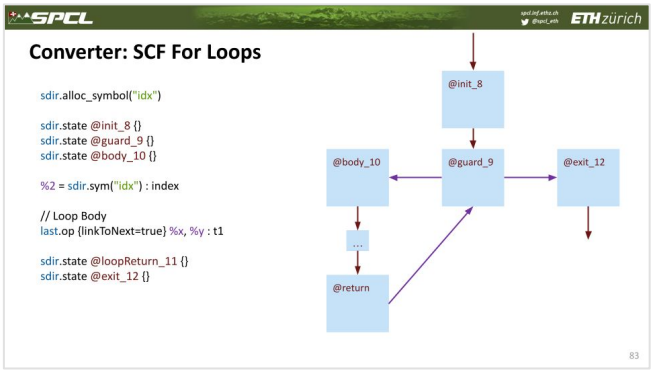
Outline

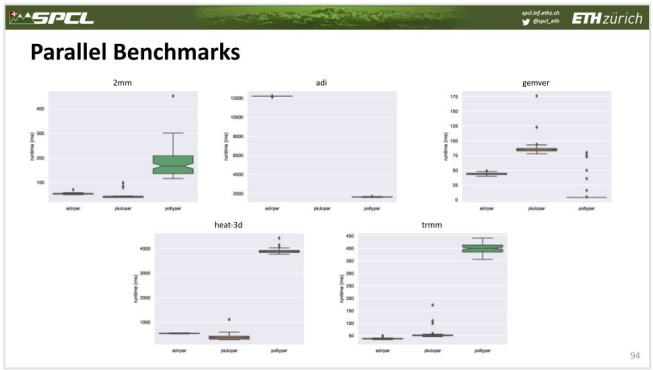
- Motivation
- SDIR
- Evaluation
- Summary

Summary


SDIR: Between MLIR and DaCe
Source Language
Language specific dialect
SDIR
MLIR
STD
Affine
Linalg
Lowered LLVM dialect
Machine Code
SDFG
Transformed SDFG
DaCe
Generated Code
42


SDIR: Consume Scopes
%A = sdir.alloc_stream()
%S = sdir.get_access %A
sdir.consume{num_pes=5, condition=@empty} (%S) -> (pe: %p, elem: %e) {
%c = sdir.call @add_one(%e)
sdir.store{wcr="add"} %c, %C[0]
}
52

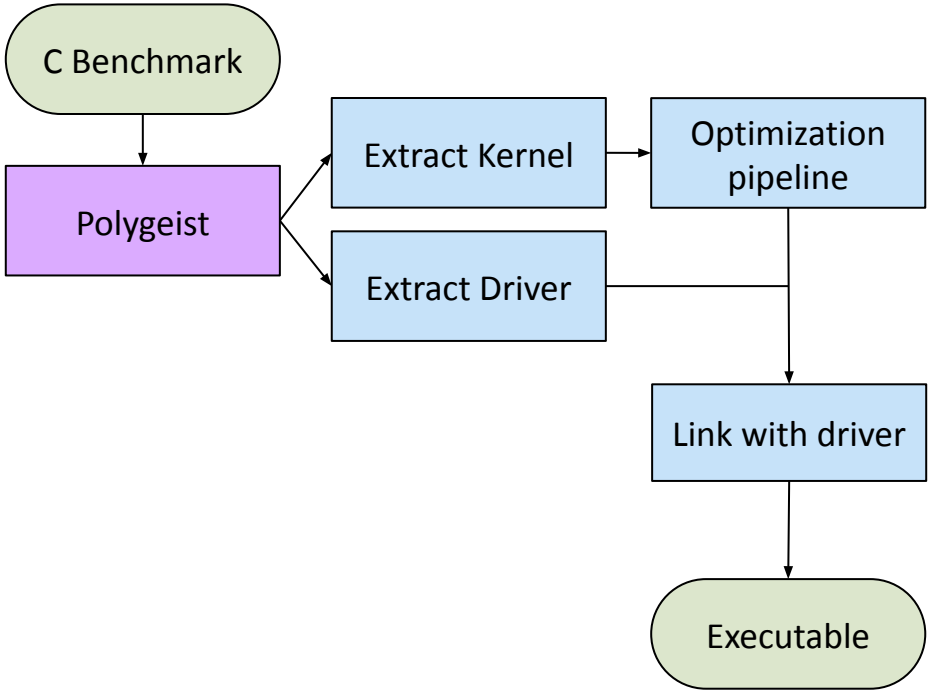

Converter: SCF For Loops
sdir.alloc_symbol("idx")
sdir.state @init_8 {}
sdir.state @guard_9 {}
sdir.state @body_10 {}
%2 = sdir.sym("idx") : index
// Loop Body
last.op {linkToNext=true} %x, %y : t1
sdir.state @loopReturn_11 {}
sdir.state @exit_12 {}
@init_8
@guard_9
@body_10
@exit_12
...
@return
83


Parallel Benchmarks
2mm
adi
gemver
heat-3d
trmm
94

Berke Ates, Prof. Dr. Torsten Hoefler, Dr. Tal Ben-Nun, Dr. Alexandru Calotoiu

SDIR: A Data-Centric Dialect for MLIR

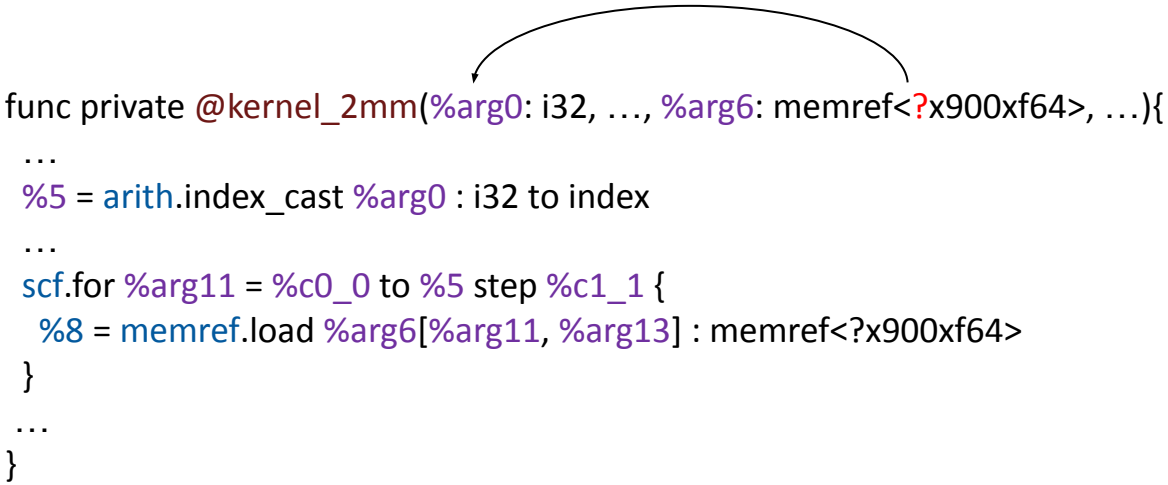
Further Evaluations



Challenges

```
func private @kernel_2mm(%arg0: i32, …, %arg6: memref<?x900xf64>, …){
  …
  %5 = arith.index_cast %arg0 : i32 to index
  …
  scf.for %arg11 = %c0_0 to %5 step %c1_1 {
    %8 = memref.load %arg6[%arg11, %arg13] : memref<?x900xf64>
  }
 …
}
```

Challenges


func private @kernel_2mm(%arg0: i32, …, %arg6: memref<?x900xf64>, …){
…
%5 = arith.index_cast %arg0 : i32 to index
…
scf.for %arg11 = %c0_0 to %5 step %c1_1 {
%8 = memref.load %arg6[%arg11, %arg13] : memref<?x900xf64>
}
…
}

Challenges

func private @kernel_2mm(%arg0: i32, …, %arg6: memref<?x900xf64>, …)

!sdir.memlet<sym("s_0")x900xf64>

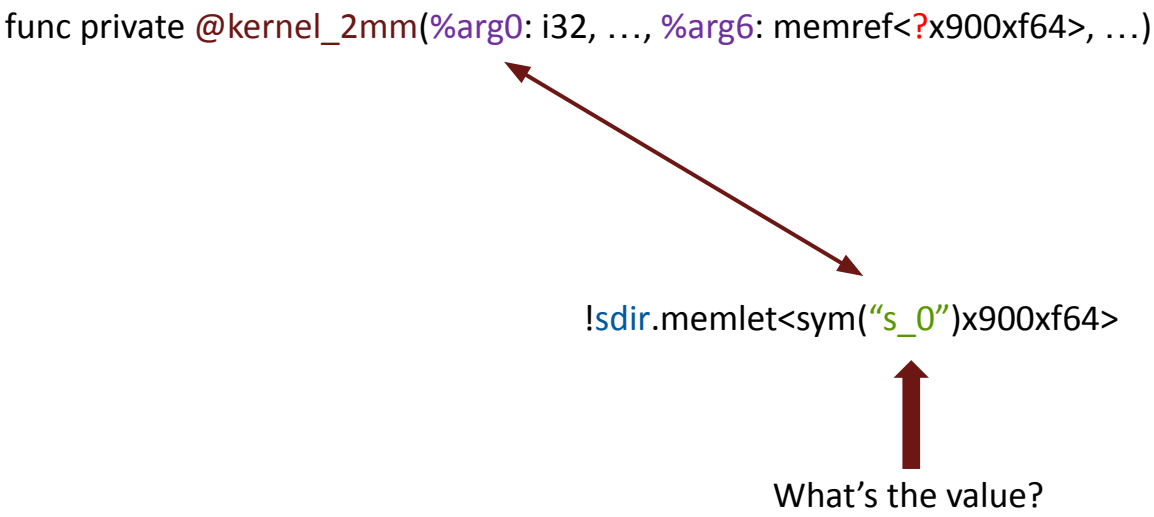
Challenges

func private @kernel_2mm(%arg0: i32, …, %arg6: memref<?x900xf64>, …)

!sdir.memlet<sym("s_0")x900xf64>

What's the value?

Challenges


func private @kernel_2mm(%arg0: i32, …, %arg6: memref<?x900xf64>, …)
!sdir.memlet<sym("s_0")x900xf64>
What's the value?