# PolyLingual: a Programmable Polyhedral Scheduler

Tom Hammer Vincent Loechner

Université de Strasbourg & Inria CAMUS team

IMPACT '23, Toulouse

# Outline

1. Context
   - ILP schedulers
   - Implement schedulers through a DSL
2. PolyLingual
   - The Periscop Toolchain
   - Structure of a PolyLingual program
     - Specification
     - Algorithm
3. Conclusion

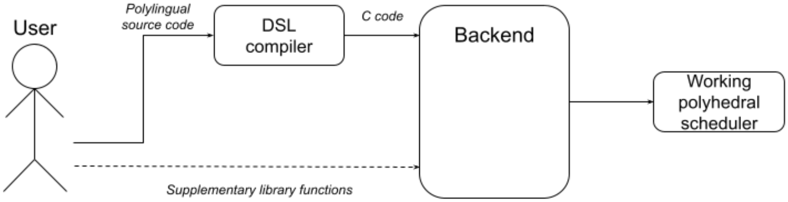
Université de Strasbourg

Inria

# ILP schedulers: Tedious to implement

- Different algorithms use the same mechanics
  - Generating constraints
  - Aggregating into ILP formulations
- Schedulers are difficult to implement
  - External libraries for scheduling related tasks
  - Complicated matrix manipulation
- What if we could implement polyhedral schedulers by transcribing their algorithms ?

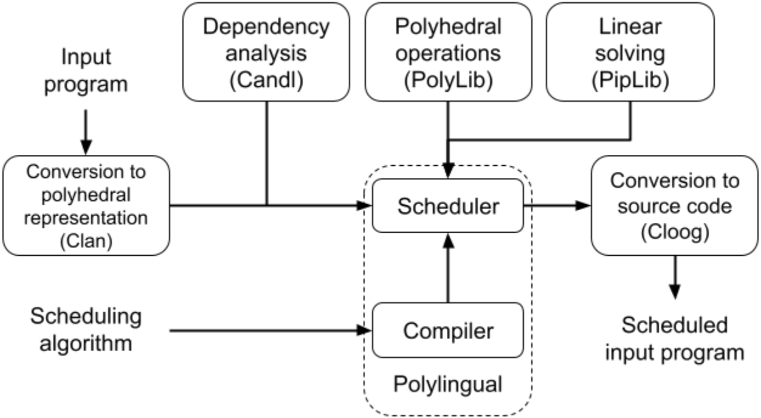
# PolyLingual: a DSL and compiler




- Specifying constraints through their mathematical formulas
- Implementing the algorithm by writing it in the DSL syntax
- Leaving room for expansion

# Outline

# The Periscop toolchain



# Program structure

- Specification of general information about the scheduler
    - Form of the schedule
    - Schedule constraints, objective functions/variables
    - Generation of information about coefficients and their respective dimensions
- Implementation of the scheduling algorithm
    - Manipulation of the input program's data
    - Formulation of an ILP
    - Function calls

# Specification identifiers

- Specifying the schedule for statements

**Specification**

$$\theta_S(\vec{\imath}, \vec{p}) = \vec{c_S^{\vec{\imath}}}.\vec{\imath} + \vec{0}.\vec{p} + c_S^c.1$$

**PolyLingual Syntax**

SCHEDULE
c_i * IT_VEC + c_c * CONS_VEC

- Specifying constraints and objective functions/variables
  - Schedule constraints
  - Objective functions
  - Objective variables

**Specification**

$$\theta_T - \theta_S \geq 0$$

**PolyLingual Syntax**

S_CONS legality =
(T_SCHED - S_SCHED >= 0)

# Generated code

## Example: Positivity constraint

$$c_i \geq 0, \sum c_i \geq 1$$

## PolyLingual Syntax

O_FUNC positivity = (c_i >= 0, SUM(c_i) >= 1)

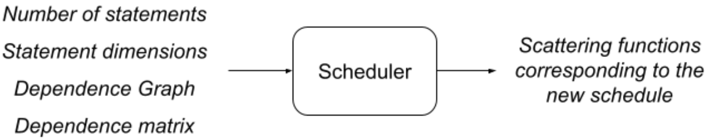
- The constraint is compiled into a C function
- The parameters passed are then later calculated depending on the ILP

## Produced function

```
Matrix *positivity(int dim){
  int i;
  Matrix *out;
  int count = 0;
  out = Matrix_Alloc(1+ (dim*1), (
       dim*1) + 2);
  for(i = 0; i < out->NbRows; i++){
    value_assign(out->p[i][0], 1);
  }
  for(i = 0; i < dim; i++){
    value_assign(out->p[count+i][i +
        (dim*0) + 1],+1);
  }
  count += dim;
  for(i = 0; i < dim; i++){
    value_assign(out->p[count][i + (
        dim*0) + 1],+1);
  }
  value_assign(out->p[count][out->
       NbColumns - 1],-1);
  count++;
  return out;
}
```

Université de Strasbourg

# Input program data




- Functions with fixed parameters
  - Statements, dependencies and DDG
- Manipulation of sets
  - Subsetting operations
  - Iterations
- Calls to library functions

# Data types

- Elementary types
  - Integer, Boolean
  - Used for conditions or indices
- Abstract types
  - Statement, Dependency, S_Set, D_Set, System, ILP, Solution, Graph.
  - All have sub-identifiers
  - New sub-identifiers may be implemented

**Example:** *System d.cons = ...*

Assigning a System to *d.cons* creates a *cons* element of type *System* in the data structure for variable *d*.
Thus, all variables of the same type now have a *cons* member.

# ILP formulation

**PolyLingual syntax for the ILP formulation of Pluto**

ILP problem = (u:global, w:global, c_i, c_c)

- Defines the order of the coefficients
- Scopes define how constraints are aggregated
  - global
  - dep
  - stmt
- Generates C functions used when calling *add_to_ILP()*
- The schedule coefficients are assigned the *stmt* scope automatically

# PolyLingual implementation of the Pluto algorithm

```
forall d in D {
  System c1 = apply_to(legality, d)
  System c2 = apply_to(
       volume_bounding, d)
  System c3 = aggregate(c1, c2)
  d.cons = c3
}
repeat {
  ILP problem = (u:global, w:global
       , c_i, c_c)
  forall d in D {
    add_to_ILP(d.cons, problem)
  }
  forall s in S {
    System c = apply_to(positivity,
         s)
    add_to_ILP(c, problem)
  }
  Solution sol = solve(problem)
  Boolean solution_found = sol.
       found
```

```
  while sol.found {
    store_schedule(sol, S)
    forall s in S {
      System orth =
          orth_schedule_space(s)
      add_to_ILP(orth)
    }
    sol = solve(problem)
  }
  if !solution_found {
    Graph DAG = gen_DAG(DDG)
    forall n in DAG {
      forall s in n.statements {
        insert_scalar_dim(n.order,
             s)
      }
    }
  }
  update()
}until (subset(D, solved == True).
     length == 0) and ~~or~~ (
     orth_schedule_space(max(D, dim
     )) == NULL)
```

# Conclusion

- A simple way to specify polyhedral schedulers
  - Mathematical syntax
  - Implementation close to the algorithm specification
  - Versatile
- Going further
  - Multi-algorithm possibilities
  - Output schedule manipulation

Université de Strasbourg

Inria

# Conclusion

- A simple way to specify polyhedral schedulers
  - Mathematical syntax
  - Implementation close to the algorithm specification
  - Versatile
- Going further
  - Multi-algorithm possibilities
  - Output schedule manipulation

**Thank you for your feedback !**