

GRAPHITE: Gimple Represented as Polyhedra

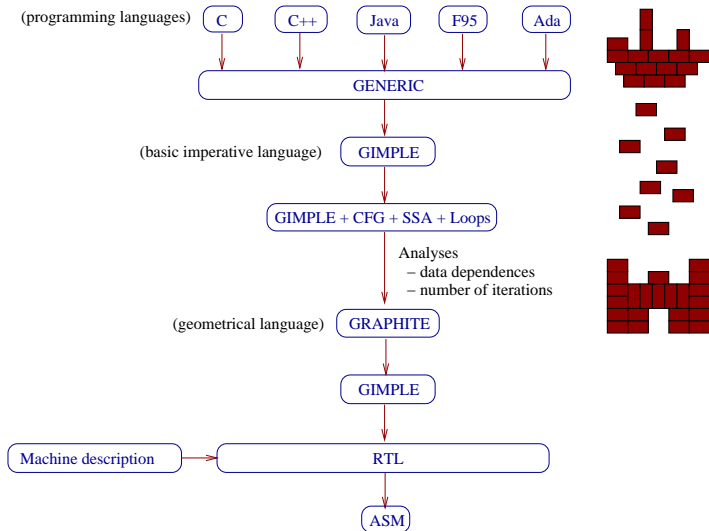
Sebastian Pop

sebastian.pop@inria.fr

INRIA Futurs - Orsay, France

2nd HiPEAC GCC Tutorial
Ghent, Belgium
January, 2007

Architecture of GCC and Loop Nest Optimizer



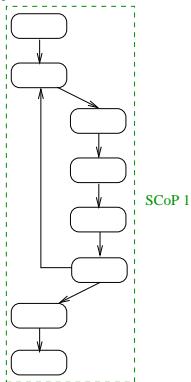
Motivations for GRAPHITE:

- difficult to undo loop transforms
- order of transforms fixed once for all
- difficult to compose
- invalidated data deps: ad-hoc correction or rebuild
- transform applied to loop bodies

GRAPHITE built on top of:

- scalar evolutions: number of iterations, access functions
- array and pointer analyses
- data dependence analysis
- scalar range estimations: undefined signed overflow, undefined access over statically allocated data, etc.

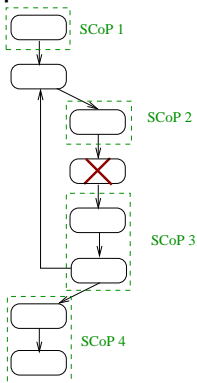
- SCoPs built on top of the CFG:



basic blocks of the SCoP

- contain only affine constructs, no side effects

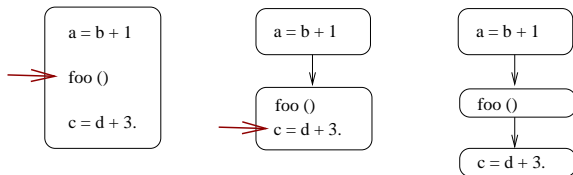
- SCoPs built on top of the CFG:



basic blocks of the SCoP

- contain only affine constructs, no side effects
- dominated by entry, postdominated by exit of the SCoP

- SCoPs built on top of the CFG:
- splitting basic blocks: `split_block`



basic blocks = containers
split basic blocks for:

- smaller code chunks
- reduce number of dependences
- moving parts of code around

- parameters = variables varying outside a SCoP
 - function parameters
 - variables varying in outer loops
- context = constraints on parameters
 - use IPA info for bounding function parameters
 - use VRP's propagation engine

Statements + parametric affine inequalities

- 1 a **domain** = bounds of enclosing loops

```
for (i=0; i<m; i++)  
  for (j=5; j<n; j++)  
    A[2*i][j+1] = ...
```

$$\begin{bmatrix} i & j & m & n & cst \\ \hline 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 5 \\ 0 & -1 & 0 & 1 & -1 \end{bmatrix}$$

$$\begin{aligned} i &\geq 0 \\ -i + m &\geq -1 \\ j &\geq 5 \\ -j + n &\geq -1 \end{aligned}$$

Statements + parametric affine inequalities

- 1 a **domain** = bounds of enclosing loops
 - 2 a list of **access functions**
-

```
for (i=0; i<m; i++)  
  for (j=5; j<n; j++)  
    A[2*i][j+1] = ...
```

$$\left[\begin{array}{ccccc} i & j & m & n & cst \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array} \right] \quad \begin{array}{l} 2 * i \\ j + 1 \end{array}$$

Statements + parametric affine inequalities

- 1 a **domain** = bounds of enclosing loops
 - 2 a list of **access functions**
 - 3 a **schedule** = execution time (static + dynamic)
-

- sequence $[[s_1; s_2]]$:

$$S[[s_1]] = t$$

$$S[[s_2]] = t + 1$$

- loop $[[loop_1 \ s \ end_1]]$: i_1 indexes $loop_1$ iterations: dynamic time

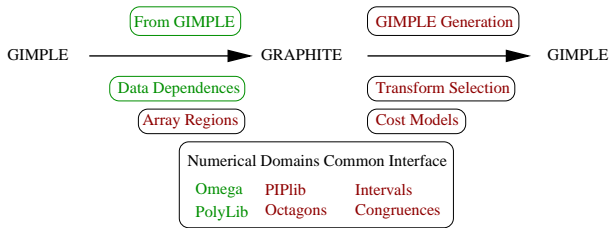
$$S[[loop_1]] = t$$

$$S[[s]] = (t, i_1, 0)$$

- 1 fill CLooG program structures: context, statement domains, scheduling functions
- 2 `cloog_program_generate`
- 3 `cloog_clast_create`
- 4 walk the AST, recompose a GIMPLE program:
 - modify loop structure
 - `create_iv`
 - reuse parts of `lambda-code.c`

GRAPHITE: Road Map

- select SCoPs
- lib integration PolyLib, CLooG, PiPLib, Omega
- extend lambda-code.c interface with CLooG
- cost models more static analyzers, and transform selection
- array regions improve data deps in interproc mode



Questions?