Toward a Polynomial Model, Season III

Polynomial Code Generation

Paul Feautrier[1] Albert Cohen[2] Alain Darte[3]

[1]Ecole Normale Supérieure de Lyon

[2]Equipe INRIA Parkas

[3]Xilinx and CNRS

January 15, 2018

Polynomials Everywhere

- The polyhedral model deals only with affine forms i.e. polynomials of degree one.
- Polynomials are needed:
  - If present in the source e.g. when computing distances
  - After evalation of induction variables
  - After linearization of arrays
  - When counting messages, operations, memory cells ....

Mathematical Background

Needed: an equivalent of Farkas lemma for building positivity certificates.

- Semi-algebraic sets:

$$S = \{x \in R^n | p_i(z) \geq 0, i = 1, n\}$$

  where the p_i are *p*olynomials.

- Theorems by Handelman, Schweighofer, Putinar:
- Schweighofer products:

$$g_\alpha(x) = p_1(x)^{\alpha_1} ... p_n(x)^{\alpha_n}.$$

- $P(x)$ is stricly positive in S iff it is a positive linear combination of Schweighofer products
- Minor conditions: S must be compact and the g_i must generate all polynomials.
- Note that there is no integral version of these theorems.

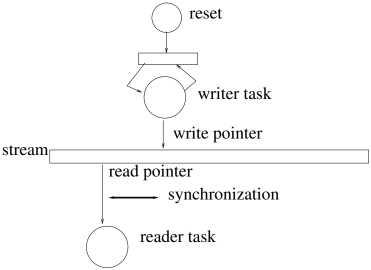
Mechanics

Expand the master equation:

$$P(x) = \sum_{\alpha} \lambda_\alpha g_\alpha, \ \lambda_\alpha \geq 0,$$

- Equate coefficients of like monomials
- The result is a linear system of equations in the λs to be solved in positive unknowns by any linear program solver.
- Linear solvers are very powerful and can tackle problems with thousands of constraints and unknowns.
- Since one must limit the number of Schweighofer products, the problem is only semi-decidable.

The OpenStream Language



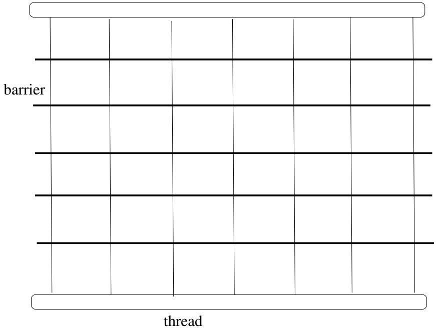

```
stream s, t;
task reset{
  write once into s;   //theta() = 0
}

for(i=0;;i++)
  task writer{         //theta(i) = i+1
    read once from s;
    write once into s;
    write once into t;
  }

for(i=0;; i++)
  task reader{         //theta(i) = i+2
   read once from t;
  }
```

- A stream is a potentially infinite one dimensional array, with a write pointer and a read pointer.
- At each read or write, the corresponding pointer is increased by a non negative amount, the *burst*.
- The read pointer cannot overtake the write pointer : synchronization.
- Analogy with Unix files and hardware channels.

Dependences and Scheduling

- If the control program is polyhedral, one can obtain closed form formulas for pointers by counting task creations using ISCC. The results are polynomials, hence the dependence relation is semi-algebraic.
- One can obtain polynomial schedules using Handelman or Schweighofer theorems.
- See IMPACT 2015, 2016.

Code Generation Basics



- Each thread execute sequentially all instances of one task.
- After each instance, the thread execute some `barriers`.
- The number of barriers from the begining of the stream to a given instance must be equal to the schedule of the instance.

The Problem of the Decreasing Schedule

Since the number of barriers can only increase, task instances must be created in order of increasing schedule. Let $\ll$ be the execution order of the control program, and θ be the schedule of a task.

- If the system of constraints

  $$u \ll v, \theta(v) < \theta(u)$$

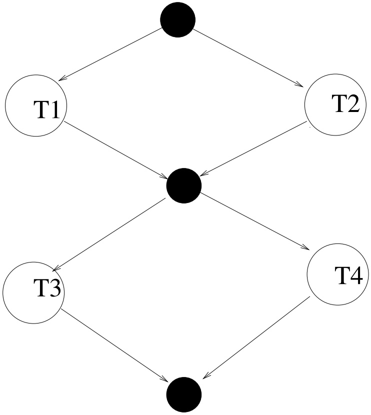
  is unfeasible, the schedule is increasing.
- If

  $$u \ll v, \theta(u) < \theta(v),$$

  is unfeasible, the schedule is decreasing, the execution order must be reversed.
- If both systems are feasible, the schedule is non monotonic. Index set splitting?
- If both system are unfeasible, the schedule is constant.

Target Languages: X10 / Habanero

```
clocked finish{
  clocked async{
    T1;
  }
  clocked async{
    T2;
  }
  clocked async{
    advance;
    T3;
  }
  clocked async{
    advance;
    T4;
  }
}
```



Related Work

- Counting Algorithms: Barvinok, Brion, Clauss and the Strasburg school, Ehrhart, Verdoolaege. Note that to the best of my knowledge, there is no equivalent for semi-algebraic sets.
- Delinearization, CART, CRP: avoiding polynomials.
- Achtziger and Zimmerman on quadratic schedules.
- Groesslinger on cylindrical algebraic decomposition.
- Clauss et. al. on inverting schedules.

Conclusion and Future Work

- An implementation is under way.
- Needs to be extended: data parallelism, non monotonic schedules, task body.
- OpenStream is an interesting language: hiding non polyhedral code in the task body, HLS.
- A small step beyond the polyhedral model
- Missing tools:
  - A projection algorithm (CAD ?) and a transitive closure algorithm
  - A counting algorithm
  - A polynomial version of the Cousot-Halbwachs algorithm.
- Other Models ??