# Static Analysis of OpenStream Programs

## Using polyhedral techniques to analyze interesting language subsets

Alain Darte
With Albert Cohen and Paul Feautrier

CNRS, Compsys
Laboratoire de l'Informatique du Parallélisme
École normale supérieure de Lyon

JFC'15: journées françaises de compilation
21-23 septembre 2015, Banyuls sur Mer

# Parallel languages, runtime execution, and static analysis

Solution(s) for high-level parallel programming?

- Static or dynamic?
- Language constructs or libraries?
- Expressiveness: deterministic (no data races) or deadlock-free?
- How to represent communications and memories? Concurrency?
- Can static optimization help runtime optimizations?
  Buffer sizes, granularity, mapping, ...

# Parallel languages, runtime execution, and static analysis

Solution(s) for high-level parallel programming?

- Static or dynamic?
- Language constructs or libraries?
- Expressiveness: deterministic (no data races) or deadlock-free?
- How to represent communications and memories? Concurrency?
- Can static optimization help runtime optimizations?
  Buffer sizes, granularity, mapping, . . .

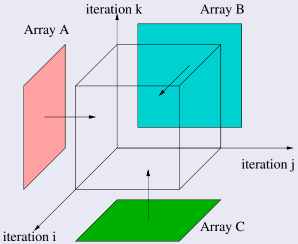
Many approaches:

- "Lower"-level: MPI, OpenCL, Lime, . . .
- Runtime-based: Kaapi, StarPU (with task dep. as in OpenMP 4.0).
- (A)PGAS languages: Co-Array Fortran, UPC, Chapel, X10, . . .
- "Dataflow" languages: KPN, SDF, CSDF, SigmaC, OpenStream, . . .

# Multi-dimensional affine representation of loops and arrays

## Matrix Multiply

```
int i,j,k;
for(i = 0; i < n; i++) {
   for(j = 0; j < n; j++) {
S:     C[i][j] = 0;
       for(k = 0; k < n; k++) {
T:         C[i][j] += A[i][k] * B[k][j];
       }
   }
}
```




## Polyhedral Description — Omega/ISCC syntax

```
Domain := [n]->{S[i][j]: 0<=i,j<n; T[i][j][k]: 0<=i,j,k<n};

Read  := [n]->{T[i][j][k]->A[i][k]; T[i][j][k]->B[k][j];
               T[i][j][k]->C[i][j]};

Write  := [n]->{S[i][j]->C[i][j]; T[i][j][k]->C[i][j]};

Order  := [n]->{S[i][j]->[i][j][0]; T[i][j][k]->[i][j][1][k]};
```

# Limits of automation: often related to the polyhedral model

Polyhedral model: ambiguity with the word “model”.

- Programming/specification model: ☛ Feautrier’s model, Alpha, CRP
- Set of provable techniques under some hypotheses: ☛ SCoP.
- Simplified specification for extracting more general facts.
  ☛ Principle: study a polyhedral subset of a specification/language.

# Limits of automation: often related to the polyhedral model

Polyhedral model: ambiguity with the word "model".

- Programming/specification model: ☛ Feautrier's model, Alpha, CRP
- Set of provable techniques under some hypotheses: ☛ SCoP.
- Simplified specification for extracting more general facts.
  ☛ Principle: study a polyhedral subset of a specification/language.

Examples:

- Uniform loops (simple case to discuss NP-completeness).
- Polyhedral X10 (Yuki, Feautrier, Rajopadhye, Saraswat, PPoPP'13).
- Polyhedral OpenStream (this talk).

# Analyzing X10 through a polyhedral fragment

X10 language developed at IBM, variant at Rice (V. Sarkar)

- PGAS (partitioned global address space) memory principle.
- Parallelism of threads: in particular keywords finish, async, clock.
- No deadlocks by construction but non-determinism.

Polyhedral X10 Yuki, Feautrier, Rajopadhye, Saraswat (PPoPP 2013)

Can we analyze the code for data races?

```
finish {
  for(i in 0..n-1) {
    S1;
    async {
      S2;
    }
  }
}
```

```
clocked finish {
  clocked async {
    S1;
    advance();
  }
  S2;
  advance();
}
```

# Polyhedral X10 dependence analysis

Let $W : A[f(\vec{x})] = \ldots$ be a write and $R : \ldots = A[g(\vec{y})]$ a read in array $A$.

Imperative sequential code (total order)

- $\prec$: disjunction of affine conjunctions (lexicographic order).
- Find the maximal $\vec{x}$, w.r.t. $\prec$, such that:
  - $\vec{x}$ and $\vec{y}$ are legal iterations $\vec{x} \in D_W$; $\vec{y} \in D_R$.
  - $W$ and $R$ access the same element of $A$: $f(\vec{x}) = g(\vec{y})$.
  - $\vec{x} \prec \vec{y}$.

☛ Maximization, gives exact dependence analysis (Feautrier IJPP 1991).

# Polyhedral X10 dependence analysis

Let $W : A[f(\vec{x})] = \ldots$ be a write and $R : \ldots = A[g(\vec{y})]$ a read in array $A$.

Imperative sequential code (total order)

- $\prec$: disjunction of affine conjunctions (lexicographic order).
- Find the maximal $\vec{x}$, w.r.t. $\prec$, such that:
  - $\vec{x}$ and $\vec{y}$ are legal iterations $\vec{x} \in D_W$; $\vec{y} \in D_R$.
  - $W$ and $R$ access the same element of $A$: $f(\vec{x}) = g(\vec{y})$.
  - $\vec{x} \prec \vec{y}$.

☛ Maximization, gives exact dependence analysis (Feautrier IJPP 1991).

Polyhedral X10 fragment (add just async: partial order)

- $\prec$ can still be expressed as an incomplete lexicographic order.
- $\vec{x} \prec \vec{y}$ becomes $\neg(\vec{y} \prec \vec{x})$ and $\vec{x} \neq \vec{y}$.
- Maximization now builds a set of (non-comparable) extrema.
- W/W and R/W races can be checked similarly.

☛ Yuki, Feautrier, Rajopadhye, Saraswat (PPoPP 2013).

# What about X10 clocks?

Assume a single clock

$\phi(\vec{x})$ “phase” number of $\vec{x}$: number of advances before $\vec{x}$.

➢ Depends only on $\prec$: can be expressed by polyhedral techniques.

Also, $\vec{x} \prec \vec{y} \Rightarrow \phi(\vec{x}) \leq \phi(\vec{y})$.

Clocks adds ordering constraints

➢ New partial order $\prec_c$: $\vec{x} \prec_c \vec{y}$ iff $\vec{x} \prec \vec{y}$ or $\phi(\vec{x}) < \phi(\vec{y})$.

Detection of races?

➢ Now becomes undecidable (Hilbert 10th problem), Feautrier (2013).

X10: expressiveness + deadlock-free $\Rightarrow$ non-determinism + undecidability

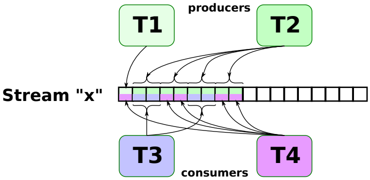
OpenStream: expressiveness + determinism $\Rightarrow$ deadlocks + undecidability.

# Analyzing OpenStream through a polyhedral fragment

```
#pragma omp task output (x) // Task T1
x = ...;
for (i = 0; i < N; ++i) {
  int window_a[2], window_b[3];

  #pragma omp task output (x « window_a[2]) // Task T2
  window_a[0] = ...; window_a[1] = ...;
  if (i % 2) {
    #pragma omp task input (x » window_b[2]) // Task T3
    use (window_b[0], window_b[1]);
  }
  #pragma omp task input (x) // Task T4
  use (x);
}
```

(Pop, Cohen, 2011)



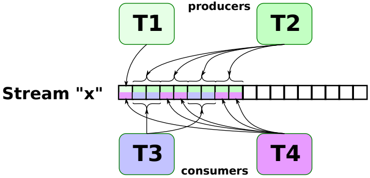

- Sequential control program for task activations.
- Reservation for reads/writes in streams with burst and horizon.
- Single assignment in streams (by construction) + dataflow semantics.

# Analyzing OpenStream through a polyhedral fragment

```
#pragma omp task output (x) // Task T1
x = ...;
for (i = 0; i < N; ++i) {
  int window_a[2], window_b[3];

  #pragma omp task output (x << window_a[2]) // Task T2
  window_a[0] = ...; window_a[1] = ...;
  if (i % 2) {
    #pragma omp task input (x >> window_b[2]) // Task T3
    use (window_b[0], window_b[1]);
  }
  #pragma omp task input (x) // Task T4
  use (x);
}
```

(Pop, Cohen, 2011)




- Sequential control program for task activations.
- Reservation for reads/writes in streams with burst and horizon.
- Single assignment in streams (by construction) + dataflow semantics.

- Instance of Pop-Cohen CDDF (Control-Driven Data Flow).
- Optimization in Erbium runtime system explored by Pop & Miranda.
- Unlike KPN, streams with multiple inputs/outputs (but deterministic).

# Some properties of polyhedral OpenStream

- The order of activations is the sequential order of the control program.
- Access functions to streams are statically-expressible polynomials.
- Dependence analysis is feasible with tools manipulating polynomials.

# Some properties of polyhedral OpenStream

- The order of activations is the sequential order of the control program.
- Access functions to streams are statically-expressible polynomials.
- Dependence analysis is feasible with tools manipulating polynomials.

Deadlock characterization iff there is, in the unrolled dependence graph,

- a statement $T$ and an infinite sequence $(i_j)_{j \in N}$ such that $T(i_j)$ depends on $T(i_{j+1})$ and $i_j$ is before $i_{j+1}$ in the activation program;
- a cycle if the program is bounded or follows Kahnian semantics.

# Some properties of polyhedral OpenStream

- The order of activations is the sequential order of the control program.
- Access functions to streams are statically-expressible polynomials.
- Dependence analysis is feasible with tools manipulating polynomials.

Deadlock characterization iff there is, in the unrolled dependence graph,

- a statement $T$ and an infinite sequence $(i_j)_{j \in N}$ such that $T(i_j)$ depends on $T(i_{j+1})$ and $i_j$ is before $i_{j+1}$ in the activation program;
- a cycle if the program is bounded or follows Kahnian semantics.

☛ Consequence: if there is a schedule for a given size of streams, this size can be enforced (back pressure) without creating deadlocks at runtime.

# Some properties of polyhedral OpenStream

- The order of activations is the sequential order of the control program.
- Access functions to streams are statically-expressible polynomials.
- Dependence analysis is feasible with tools manipulating polynomials.

Deadlock characterization iff there is, in the unrolled dependence graph,

- a statement $T$ and an infinite sequence $(i_j)_{j \in N}$ such that $T(i_j)$ depends on $T(i_{j+1})$ and $i_j$ is before $i_{j+1}$ in the activation program;
- a cycle if the program is bounded or follows Kahnian semantics.

☛ Consequence: if there is a schedule for a given size of streams, this size can be enforced (back pressure) without creating deadlocks at runtime.

Deadlock detection If access indices can express polynomials of large degrees, deciding the presence of deadlocks is undecidable, and one can build such an OpenStream program.

☛ Consequence: Not clear how to optimize, unless with polynomial computations (cf Feautrier, with Schweighofer/Handelman theorem).

# First ingredient (Feautrier): build multivariate polynomials

$Q(x_1, \ldots, x_n)$: multivariate polynomial, nonnegative integer coefficients.

Write:

- $Q(x) = Q(x_1, x_r)$, $x_1$ first variable.
- $Q^1(x_1, x_r) = Q(x_1 + 1, x_r) - Q(x_1, x_r)$ (first difference)
  ☛ smaller degree, still nonnegative integer coefficients.

# First ingredient (Feautrier): build multivariate polynomials

$Q(x_1, \ldots, x_n)$: multivariate polynomial, nonnegative integer coefficients.

Write:

- $Q(x) = Q(x_1, x_r)$, $x_1$ first variable.
- $Q^1(x_1, x_r) = Q(x_1 + 1, x_r) - Q(x_1, x_r)$ (first difference)
  - ☛ smaller degree, still nonnegative integer coefficients.
  - ☛ Can compute $Q(x)$ with:

```
phi = Q(0,x_r);
for (i = 0; i < x; i++) {
  phi += Q1(i, x_r);
}
```

# First ingredient (Feautrier): build multivariate polynomials

$Q(x_1, \ldots, x_n)$: multivariate polynomial, nonnegative integer coefficients.

Write:

- $Q(x) = Q(x_1, x_r)$, $x_1$ first variable.
- $Q^1(x_1, x_r) = Q(x_1 + 1, x_r) - Q(x_1, x_r)$ (first difference)
  - ☛ smaller degree, still nonnegative integer coefficients.
  - ☛ Can compute $Q(x)$ with:

```
phi = Q(0,x_r);
for (i = 0; i < x; i++) {
  phi += Q1(i, x_r);
}
```

- Keep going until $x_1$ disappears.

```
phi = Q(0,x_r);
for (i = 0; i < x; i++) {
// phi += Q1(i, x_r);
   phi += Q1(0, x_r);
   for (j = 0; j < i; j++) {
      phi += Q2(j, x_r);
   }
}
```

# First ingredient (Feautrier): build multivariate polynomials

$Q(x_1, \ldots, x_n)$: multivariate polynomial, nonnegative integer coefficients.

Write:

- $Q(x) = Q(x_1, x_r)$, $x_1$ first variable.
- $Q^1(x_1, x_r) = Q(x_1 + 1, x_r) - Q(x_1, x_r)$ (first difference)
  - ☛ smaller degree, still nonnegative integer coefficients.
  - ☛ Can compute $Q(x)$ with:

```
phi = Q(0,x_r);
for (i = 0; i < x; i++) {
  phi += Q1(i, x_r);
}
```

- Keep going until $x_1$ disappears.

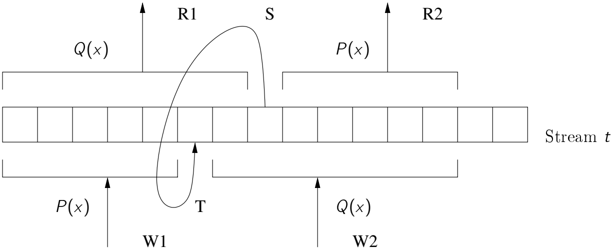
```
phi = Q(0,x_r);
for (i = 0; i < x; i++) {
// phi += Q1(i, x_r);
   phi += Q1(0, x_r);
   for (j = 0; j < i; j++) {
      phi += Q2(j, x_r);
   }
}
```

- Continue with other variables:

```
phi = Q(0,x_r); // Put new loops
for (i = 0; i < x; i++) {
// phi += Q1(i, x_r);
   phi += Q1(0, x_r); // Put new loops
   for (j = 0; j < i; j++) {
      phi += Q2(j, x_r); // Put new loops
   }
}
```

# Second ingredient: build the OpenStream structure

```
s, t streams;
for (x in D) {
/* D is the n-dimensional cube
   of size N in the first orthant */
  R1: read Q(x) times in t;
  W1: write P(x) times in t;
  S: read once in t and write once in s;
  T: read once in s and write once in t;
  R2: read P(x) times in t;
  W2: writes Q(x) times in t;
}
```

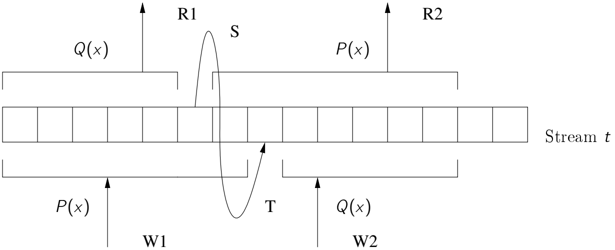



Deadlock situations:

- General case: iff $P(x) = Q(x)$.
- Kahnian case: iff $P(x) \leq Q(x)$.
  Note: iff no **causal** schedule.

# Second ingredient: build the OpenStream structure

```
s, t streams;
for (x in D) {
/* D is the n-dimensional cube
   of size N in the first orthant */
  R1: read Q(x) times in t;
  W1: write P(x) times in t;
  S: read once in t and write once in s;
  T: read once in s and write once in t;
  R2: read P(x) times in t;
  W2: writes Q(x) times in t;
}
```

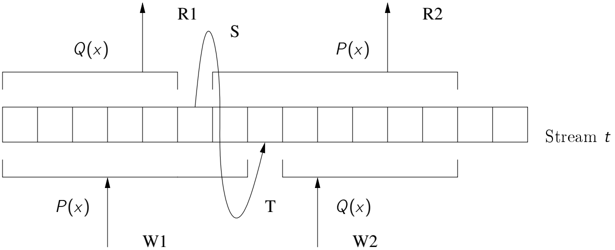



Deadlock situations:

- General case: iff $P(x) = Q(x)$.
- Kahnian case: iff $P(x) \leq Q(x)$.
  Note: iff no **causal** schedule.

# Second ingredient: build the OpenStream structure

```
s, t streams;
for (x in D) {
/* D is the n-dimensional cube
   of size N in the first orthant */
  R1: read Q(x) times in t;
  W1: write P(x) times in t;
  S: read once in t and write once in s;
  T: read once in s and write once in t;
  R2: read P(x) times in t;
  W2: writes Q(x) times in t;
}
```




Deadlock situations:

- General case: iff $P(x) = Q(x)$.
- Kahnian case: iff $P(x) \leq Q(x)$.
  Note: iff no **causal** schedule.

☛ 10th Hilbert's problem:

- $R(x) = 0$ iff $R^+(x) = R^-(x)$.
- $R(x) = 0$ iff $R^2(x) \leq 0$.

# Take-home messages

## About polyhedral specifications

- Polyhedral fragments to understand the limit of automation.
- Watch out: affine codes generate polynomials.
- Extension towards polynomial optimizations?

# Take-home messages

## About polyhedral specifications

- Polyhedral fragments to understand the limit of automation.
- Watch out: affine codes generate polynomials.
- Extension towards polynomial optimizations?

## About OpenStream and KPNs

- Emulation of one by the other not so clear.
- KPN Turing-complete because KPN includes BDF (Buck/Parks).
- But model can react on values in streams (not here).

# Take-home messages

## About polyhedral specifications

- Polyhedral fragments to understand the limit of automation.
- Watch out: affine codes generate polynomials.
- Extension towards polynomial optimizations?

## About OpenStream and KPNs

- Emulation of one by the other not so clear.
- KPN Turing-complete because KPN includes BDF (Buck/Parks).
- But model can react on values in streams (not here).

## About parallel languages

- What do you prefer: deadlocks or races?
- How to express link between user/compiler and compiler/runtime?
- Parallel constructs can help dependence analysis (cf V. Sarkar).

☛ Towards the analysis of parallel languages, with better user/compiler and compiler/runtime interactions.