Polyhedral Compilation and the Integer Set Library

Sven Verdoolaege

Cerebras Systems

(KU Leuven, LIACS, INRIA, ENS, Polly Labs)

July 4, 2022
Outline

1. Motivation (Cerebras)

2. Polyhedral Compilation

3. Integer Set Library (isl)
   - Interface
   - Internal Representation and Parametric Integer Programming
   - Operations

4. Conclusion
Cerebras Wafer-Scale Engine (WSE-2)

The Largest Chip in the World

850,000 cores optimized for sparse linear algebra
46,225 mm² silicon
2.6 trillion transistors
40 Gigabytes of on-chip memory
20 PByte/s memory bandwidth
220 Pbit/s fabric bandwidth
7nm process technology

Cluster-scale acceleration on a single chip
Automatic Code Generation

Given

• high-level algorithm description
• size of PE rectangle
• description of input and output

generate low-level (C) code exploiting hardware features

• powerful SIMD engine
• filtering
• FIFOs
• ...

⇒ Cerebras DTG tool

(for kernels for which no hand-written code is available)
Automatic Code Generation

```c
lair MV<T=float16>(M, N): T W[M][N], T x[N] -> T y[M] {
    all (i, j) in (M, N)
    y[i] += W[i][j] * x[j]
}
```

Mapping of $32 \times 16$ matrix vector multiplication to $4 \times 4$ PEs.

- **Size:** \{ PE[4, 4] \}
- **Compute Map:** \{ MV[i, j] -> PE[j//4, i//8] \}
- **I-Port Map:** \{ x[i=0:15] -> [PE[i//4, -1] -> index[i%4]] \}
- **O-Port Map:** \{ y[i=0:31] -> [PE[4, i//8] -> index[i%8]] \}
Affine Constraints

Computation instances, tensor elements, PE coordinates, ordering
⇒ represented by a tuple of integers

• Set of computation instances
  ⇒ rectangle of fixed size
  \[ \{ \text{MV}[i,j] : 0 \leq i < M \land 0 \leq j < N \} \]

• Accesses
  ⇒ affine in instance identifiers
  \[ \{ \text{MV}[i,j] \rightarrow x[j] \} \cup \{ \text{MV}[i,j] \rightarrow y[i] \} \cup \{ \text{MV}[i,j] \rightarrow W[i,j] \} \]

• Placement
  ⇒ quasi affine (may involve integer divisions)
  \[ \{ \text{MV}[i,j] \rightarrow \text{PE}[[j/4], [i/8]] \} \]

• Communication
  ⇒ quasi affine
  \[ \{ x[i = 0:15] \rightarrow \text{[PE}[[i/4], -1 \rightarrow \text{index}[i \mod 4]] \} \]

Sets and relations of integer tuples bounded by (quasi) affine constraints
Code Generation Process

Decision process involves questions of the form

- which tensor elements are needed on which PEs?
- which tensor elements are computed on which PEs?
- which computation instances can be performed on the arrival of a tensor element?
- do these computation instances form a box?
- can they be approximated by a box?
- …

Manipulation of sets and relations of integer tuples bounded by (quasi) affine constraints

⇒ Polyhedral Compilation
Polyhedral Compilation

Analyzing and/or transforming programs using the polyhedral model

Polyhedral Model

Abstract representation of a program

- instance based
  - statement instances
  - array elements
- compact representation based on polyhedra or similar objects
  - integer points in unions of parametric polyhedra
  - Presburger sets and relations
- parametric
  - description may depend on constant symbols
Polyhedral Model

Typical constituents of program representation

- **Instance Set**
  - the set of all statement instances

- **Access Relations**
  - the array elements accessed by a statement instance

- **Dependences**
  - the statement instances that depend on a statement instance

- **Schedule**
  - the relative execution order of statement instances
Illustrative Example: Matrix Multiplication

```c
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
        S1: C[i][j] = 0;
        for (int k = 0; k < K; k++)
            S2: C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- **Instance Set** (set of statement instances)

  \{ S1[i,j] : 0 \leq i < M \land 0 \leq j < N; S2[i,j,k] : 0 \leq i < M \land 0 \leq j < N \land 0 \leq k < K \} 

- **Access Relations** (accessed array elements; \( W \): write, \( R \): read)

  \begin{align*}
  W &= \{ S1[i,j] \rightarrow C[i,j]; S2[i,j,k] \rightarrow C[i,j] \} \\
  R &= \{ S2[i,j,k] \rightarrow C[i,j]; S2[i,j,k] \rightarrow A[i,k]; S2[i,j,k] \rightarrow B[k,j] \}
  \end{align*}

- **Schedule** (relative execution order)

  \{ S1[i,j] \rightarrow [i,j,0,0]; S2[i,j,k] \rightarrow [i,j,1,k] \}
Presburger Sets and Relations

Examples

\{ S1[i, j] : 0 \leq i < M \land 0 \leq j < N ; \ S2[i, j, k] : 0 \leq i < M \land 0 \leq j < N \land 0 \leq k < K \} \\
\{ S1[i, j] \rightarrow C[i, j] ; \ S2[i, j, k] \rightarrow C[i, j] \} \\

General form

- Sets

\{ S_1[i] : f_1(i) ; \ S_2[i] : f_2(i) ; \ldots \}, \\
with \( f_k \) Presburger formulas

\Rightarrow \text{ set of elements of the form } S_1[i], \text{ one for each } i \text{ satisfying } f_1(i), \ldots

- Binary relations

\{ S_1[i] \rightarrow T_1[j] : f_1(i, j) ; \ S_2[i] \rightarrow T_2[j] : f_2(i, j) ; \ldots \} \\
\Rightarrow \text{ set of pairs of elements of the form } S_1[i] \rightarrow T_1[j]

Note: despite "\( \rightarrow \)", not necessarily (single valued) functions
Quasi-affine Expressions and Presburger Formulas

- quasi-affine expression (no multiplication; only constant functions)
  - variable
  - constant integer number
  - constant symbol
  - addition (+), subtraction (−)
  - integer division by integer constant $d\ (\lfloor \cdot / d \rfloor)$

- Presburger formula
  - true
  - quasi-affine expression
  - less-than-or-equal relation ($\leq$)
  - equality ($=$)
  - first order logic connectives: $\land$, $\lor$, $\neg$, $\exists$, $\forall$

- not allowed: multiplication, functions with arity greater than zero
  - $x \times x$, $x \times N$, $f(x)$

- allowed: repeated addition
  - $3 \times x \equiv x + x + x$
Presburger Sets and Relations

General form

- Sets

\[
\{ S_1[i] : f_1(i); S_2[i] : f_2(i); \ldots \},
\]

where \( f_k(i) \) are Presburger formulas with \( i \) as only free variables

\[ \Rightarrow \] set of elements of the form \( S_1[i] \), one for each \( i \) such that \( f_1(i) \) is true, ...

Note: may depend on interpretation of symbolic constants

\[
\{ S[i] : 0 \leq i \leq n \}
\]

is equal to

\[
\begin{cases}
\emptyset & \text{if } n < 0 \\
\{ S[0] \} & \text{if } n = 0 \\
\{ S[0]; S[1] \} & \text{if } n = 1 \\
\{ S[0]; S[1]; S[2] \} & \text{if } n = 2 \\
\ldots & \\
\end{cases}
\]
Overview of isl

isl is a thread-safe C library for manipulating integer sets and relations
- bounded by affine constraints
- involving symbolic constants and
- existentially quantified variables

plus quasi-affine and quasi-polynomial functions on such domains

Supported operations by core library include
- intersection
- union
- set difference
- integer projection
- coalescing
- closed convex hull

sampling, scanning
- integer affine hull
- lexicographic optimization
- transitive closure (approx.)
- parametric vertex enumeration
- bounds on quasi polynomials

Polyhedral compilation library
- schedule trees
- dataflow analysis

- scheduling
- AST generation
Connection with other Libraries and Tools

isl: manipulates parametric affine sets and relations
barvinok: counts elements in Presburger sets and relations
pet: extracts polyhedral model from clang AST
PPCG: Polyhedral Parallel Code Generator
iscc: interactive calculator

Licenses:
BSD/MIT/
Apache
LGPL
GPL
Set Representation

S: \[ A[0] = 1; \]
   \[ \text{for } (i = 1; i < N; ++i) \]

T: \[ A[i] = 2 \times A[i - 1]; \]

- **isl**: named (and nested) spaces
  \[ [N] \rightarrow \{ S[], T[i]: 1 \leq i < N \} \]

- Omega:
  symbolic \( N \);
  \[ \{ [0, 0] \} \cup \{ [1, i]: 1 \leq i < N \} \]

- PolyLib:
  (deals with rational sets, polyhedra)

\[
\begin{align*}
2 \\
2 & 5 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
3 & 5 & \text{equality/inequality} & N \\
0 & 1 & 0 & 0 & -1 \\
1 & 0 & 1 & 0 & -1 \\
1 & 0 & -1 & 1 & -1
\end{align*}
\]
Spaces
Recall general form

- Sets

\[ \{ S_1[i] : f_1(i); S_2[i] : f_2(i); \ldots \} , \]

- Binary relations

\[ \{ S_1[i] \rightarrow T_1[j] : f_1(i,j); S_2[i] \rightarrow T_2[j] : f_2(i,j); \ldots \} \]

Tuple space:
- the identifier (e.g., \( S_1, S_2, T_1, T_2 \)), combined with
- the size, i.e., the number of elements in the tuple (e.g., \( i, j \))

A statement \( S_2[i] = T_1[j] \) means
- the identifiers \( S_2 \) and \( T_1 \) are the same, and
- the sizes of \( i \) and \( j \) are the same

Examples: \( S[] \neq S[i], S[a] = S[b], S[] \neq T[] \)
## Nested Relations

isl currently supports

- sets

\[
\{ S_1[i] : f_1(i); S_2[i] : f_2(i); \ldots \},
\]

- binary relations

\[
\{ S_1[i] \rightarrow T_1[j] : f_1(i,j); S_2[i] \rightarrow T_2[j] : f_2(i,j); \ldots \}
\]

but not

- n-ary relations

\[
\{ A[i] \rightarrow B[j] \rightarrow C[k] \rightarrow \ldots \}
\]

However, **nested** relations are supported

For example: statement instance specific memory map

\[
\{ [S[i,j] \rightarrow A[i]] \rightarrow \text{Mem}[i,j] \}
\]

In some cases, there is no clear binary decomposition and a real n-ary relation would be useful
Polyhedral Objects

Order → Schedule → Statement
Processor

Access → Storage → Memory
Array

Dependence

+ many more
Polyhedral Compiler and Types

A sufficiently advanced polyhedral compiler needs to handle many kinds of polyhedral objects.

This can cause confusion:
- exactly what kind of object does this function expect?
- does this operation on these objects make sense?

In statically typed languages (such as C++)
⇒ use types

In PolyLib, every set or binary relation is represented by a Polyhedron.
⇒ no differentiation at compile time
⇒ even at run time, only dimensionality can be checked

In Omega, every set or binary relation is represented by a Relation.
⇒ no differentiation at compile time
⇒ at run time, differentiation between tuple size(s) as well as between
  ▶ sets, and
  ▶ binary relations
Types Offered by Plain C++ Interface to isl

In isl, every set is represented by an `isl::set` or an `isl::union_set` and every binary relation is represented by an `isl::map` or an `isl::union_map`.

⇒ differentiation between sets and binary relations at compile time
⇒ at run time, differentiation between tuple size(s) and tuple name(s) (for `isl::set` and `isl::map`)

\[
\{ \text{S2}[i, j, k] : 0 \leq i < M \text{ and } 0 \leq j < N \text{ and } 0 \leq k < K \} \\
\{ \text{S1}[i, j] \rightarrow \text{C}[i, j] \}
\]

`isl::union_set` and `isl::union_map` objects may contain elements with different tuple sizes and/or names.

\[
\{ \text{S1}[i, j] \rightarrow \text{C}[i, j]; \text{S2}[i, j, k] \rightarrow \text{C}[i, j] \}
\]

⇒ no run-time checks
⇒ still maps statement instances to array elements
⇒ need for more fine-grained types
Types Offered by Templated C++ Interface to isl

- Template type for each plain type involving tuples
- Every type has 0 or more template parameters, one for each tuple,
- Template arguments are specified by application specifying tuple kind

For example,

```cpp
struct ST {}; // statement
struct AR {}; // array

isl::typed::map<ST, AR> access_relation;
isl::typed::map<ST, ST> dependence_relation;
```

Benefits
- compile-time checks
- documentation

Drawbacks
- increase in compilation time
- increase in binary size
Internal Representation of Sets and Relations

Each set or relation is stored as disjunction of conjunctions (with local variables)

\[ R = \bigcup_i R_i \quad \text{where} \quad R_i = \{ S[i] \rightarrow T[j] : \exists k : A_0 c + A_1 i + A_2 j + A_3 k \geq a \} \]

Each disjunct consists of

- affine equality and inequality constraints
- symbolic constants \( c \)
- local variables \( k \)
  - existentially quantified, or,
  - integer division \( k_i = \lfloor e_i / d_i \rfloor \)

Conversion to disjunction of conjunctions

\[ \neg (\exists a : f(x, a)) \rightarrow \neg f(x, g(x)) \]

⇒ determine a single value of \( a \) satisfying \( f(x, a) \) and write it as an explicit piecewise quasi affine expression \( g(x) \) of \( x \)
⇒ using parametric integer linear programming
Lexicographical Order

#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j] =

S = \{ [i,j] : 1 \leq j \leq i \leq N \}

Execution order:
[1,1], [2,1], [2,2], [3,1], [3,2], [3,3], [4,1], [4,2], [4,3], [4,4] [5,1], [5,2], [5,3], [5,4], [5,5]

Lexicographical order:
\[ \mathbf{a} \prec \mathbf{b} \equiv \bigvee_{i=1}^{n} \left( a_{i} < b_{i} \land \bigwedge_{j=1}^{i-1} a_{j} = b_{j} \right) \]
⇒ smaller in first position where tuples differ
Parametric Integer Programming

Given a parametric polyhedron (no disjunction; no local variables), give a description in terms of the parameters of the lexicographically minimal (or maximal) integer point.

E.g., first/last iteration of a loop nest satisfying some constraints

Technique: dual simplex + Gomory cuts

Result:

- Subdivision of parameter domain
- For each cell in subdivision an affine expression in terms of the parameters
- May include “new parameters”

\[ q = \left\lfloor \frac{\sum_i a_i p_i + c}{d} \right\rfloor \]
Parametric Integer Programming Example

\[ R = \{ [i, j] : 0 \leq -i \leq N \land 0 \leq -j \leq -i \land 0 \leq k \leq 3N \land k = -i - 2j \} \]

\[
\text{lexmin } R = \\
\quad \text{if } k < N \\
\quad \quad [-k, 0] \\
\quad \text{else} \\
\quad \quad \text{if } 3 \left[ \frac{k + N}{2} \right] \geq 2k \\
\quad \quad \quad \left[ k - 2 \left( \frac{k + N}{2} \right), -k + \left( \frac{k + N}{2} \right) \right]
\]
Parametric Integer Programming on Presburger Sets and Relations

\[ R = \bigcup_i R_i \quad R_i = \{ S[i] \rightarrow T[j] : \exists k : A_0 c + A_1 i + A_2 j + A_3 k \geq a \} \]

- Compute \text{lexmin} \( R \)
  \[ \Rightarrow \] treat \( R_i \) as a parametric polyhedron with
  - parameters \( c \) and \( i \)
  - variables \( j \) and \( k \)
  \[ \Rightarrow \] combine results over multiple disjuncts

- Quantifier elimination
  \[ \Rightarrow \] treat \( R_i \) as a parametric polyhedron with
  - parameters \( c, i \) and \( j \)
  - variables \( k \)
Internal Structure of isl

- core
  - incremental LP solver
  - ILP solver (GBR)
  - PILP solver

- operations on sets and relations
- operations on piecewise expressions
- operations on reductions of piecewise quasi polynomials
- parametric vertex enumeration
- scheduler
The Importance of Heuristics

Heuristics are used on top of core algorithms to *avoid computation* or produce *simpler results*.

### Parametric Integer Programming

- **tighten constraints:** \( 2x - 5 \geq 0 \Rightarrow x - 3 \geq 0 \)
- **detect implicit equality constraints**
- **exploit equality constraints to reduce dimension of tableau**
- **look for variables with fixed value in terms of parameters**
  \[
  \{ [i] \rightarrow [j, k] : i - 3 \leq 4j \leq i \land j \leq k \leq j + 1 \}
  \]
  \>
  - \( j \) has fixed value \( \lfloor i/4 \rfloor \)
  \>
  - compute minimum of \( k \) in terms of \( i \) and \( j \) and plug in \( j = \lfloor i/4 \rfloor \)
  \>
  \( \Rightarrow \) avoid potentially splitting up domain
- **detect symmetries** \( \sum_i a_i x_i \leq f_j(n) \)
  \>
  - replace by \( \sum_i a_i x_i \leq u \) with \( u \leq f_j(n) \) extra parameter
  \>
  - avoid considering all orderings of \( f_j(n) \)
- **combine cells with same expression for minimum**
Choice of Internal Representation

Quantifier elimination
- \( \text{isl} \) uses \( \lfloor \cdot / d \rfloor \) function symbols for quantifier elimination (obtained from parametric integer programming)
- traditionally, divisibility predicate symbols “\( d \mid \cdot \)” used instead (e.g., Omega)

Decomposition
- \( \text{isl} \) uses disjunction of conjunctions
- tree can be alternative (e.g., obtained from parametric integer programming)
  - single constraint used to separate two groups of cells
  - forces further subdivisions

a graph?
Constraints

**isl** (like other polyhedral libraries) has explicit representation for equality constraints

- In theory, equality constraint can be represented by pair of inequality constraints

\[ f(i) = 0 \quad \Rightarrow \quad f(i) \geq 0 \land f(i) \leq 0 \]

- However, explicit equality constraint more easily exploited to reduce dimensionality

Other “redundant” types of constraints could also be useful

- disequality constraint

\[ f(i) \neq 0 \quad \iff \quad f(i) \geq 1 \lor f(i) \leq -1 \]

- lexicographic constraint

\[ a \prec b \quad \iff \quad \bigvee_{i=1}^{n} \left( a_i < b_i \land \bigwedge_{j=1}^{i-1} a_j = b_j \right) \]

⇒ adjust core algorithms or expand before applying
Piecewise Expressions

- **Integer quasi affine expression**
  ⇒ Presburger term
  That is, a term constructed from variables, symbolic constants, integer constants, addition (+), subtraction (−) and integer division by a constant (\([\cdot]/d]\))

- **Rational polynomial expression**
  ⇒ a term constructed from variables, symbolic constants, rational constants, addition (+), subtraction (−) and multiplication (\(\cdot\))

- **Quasi polynomial expression**
  ⇒ a rational polynomial expression with variables replaced by integer quasi affine expressions

- **Piecewise quasi affine/polynomial expression**
  ⇒ a list of pairs of Presburger sets and quasi affine/polynomial expressions \(E = (S_i, e_i)_i\), with \(S_i\) disjoint

\[
E(j) = \begin{cases} 
  e_i(j) & \text{if } j \in S_i \\
  \bot/0 & \text{otherwise}
\end{cases}
\]

- \([x/2] + 3N\)
- \(x^2 - N/2\)
- \((\lceil x/2 \rceil + 3N)^2 - N/2\)
Piecewise Expressions

- Piecewise quasi affine/polynomial expression
  \[ E = (S_i, e_i)_i, \text{ with } S_i \text{ disjoint} \]

  \[ E(j) = \begin{cases} 
  e_i(j) & \text{if } j \in S_i \\
  \bot/0 & \text{otherwise}
  \end{cases} \]

- Piecewise quasi affine expression *typically* represents element of set (e.g., \text{lexmin})
  \( \Rightarrow \) undefined when set is empty

- Piecewise quasi polynomial expression *typically* represents cardinality of set
  \( \Rightarrow \) zero when set is empty

**But:** faithful conversion from partially defined piecewise quasi affine expression to piecewise quasi polynomial expression is currently not possible in \text{isl}
Value Semantics

Conceptually, each isl operation produces new object, leaving inputs untouched

However, internally,

- objects are reference counted
- an operation may return (a copy of) one of its inputs
- an input with a single reference may be reused and modified for result
- representation of shared object may get changed (not meaning)
  For example,
  - redundant constraints
  - implicit equality constraints
  - coalescing
- properties are shared among copies of same object (e.g., emptiness)
Deltas

\[ R = \{ S[i] \rightarrow S[j] : P(i, j) \} \]

\[ \Delta R = \{ S[k] : \exists i, j : S[i] \rightarrow S[j] \in R \land k = j - i \} \]

Example:

\[ R = \{ S[i_1, i_2] \rightarrow S[0, j_2] : 0 \leq i_1 \leq 10 \land 0 \leq i_2 \leq 10 \land i_2 \leq j_2 \leq i_2 + 2 \} \]

\[ \Delta R = \{ S[k_1, k_2] : -10 \leq k_1 \leq 0 \land 0 \leq k_2 \leq 2 \} \]

- Elements of \( \Delta R \) live in same space as domain and range of \( R \)
- Does it make sense to intersect \( \Delta R \) with \( \text{dom} \ R \)?
- In templated interface, method only available for relations with two identical tuple kinds
  - result has same tuple kind
  - does not guarantee that tuple spaces are the same

\[ \{ S1[i_1, i_2] \rightarrow S2[j_1, j_2] : \ldots \} \]
Set Coalescing

After many applications of projection, set difference, union, a set may be represented as a union of many disjuncts ⇒ try to combine several disjuncts into a single disjunct

\[ S_1 = \{ x : Ax \geq c \} \quad S_2 = \{ x : Bx \geq d \} \]

PolyLib way:
1. Compute \( H = \text{conv.hull}(S_1 \cup S_2) \)
2. Replace \( S_1 \cup S_2 \) by \( H \setminus (H \setminus (S_1 \cup S_2)) \)

isl way:
1. Classify constraints
   - redundant: \( \min \langle a_i, x \rangle > c_i - 1 \) over remaining constraints of \( S_1 \)
   - valid: \( \min \langle a_i, x \rangle > c_i - 1 \) over \( S_2 \)
   - separating: \( \max \langle a_i, x \rangle < c_i \) over \( S_2 \); special cases:
     * adjacent to equality: \( \langle a_i, x \rangle = c_i - 1 \) over \( S_2 \)
     * adjacent to inequality: \( \langle (a_i + b_j), x \rangle = (c_i + d_j) - 1 \) over \( S_2 \)
   - cut: otherwise
Set Coalescing

Case distinction
1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
   $\Rightarrow$ $S_2$ can be dropped
Set Coalescing

Case distinction

1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
2. no separating constraints and cut constraints of $S_2$ are valid for cut facets of $S_1$

$\Rightarrow$ replace $S_1$ and $S_2$ by disjunct with all valid constraints
Set Coalescing

Case distinction
1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
2. no separating constraints and cut constraints of $S_2$ are valid for cut facets of $S_1$
3. single pair of adjacent inequalities (other constraints valid)
   $\Rightarrow$ replace $S_1$ and $S_2$ by disjunct with all valid constraints
Set Coalescing

Case distinction

1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
2. no separating constraints and cut constraints of $S_2$ are valid for cut facets of $S_1$
3. single pair of adjacent inequalities (other constraints valid)
4. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + other constraints of $S_1$ are valid
   + constraints of $S_2$ valid for facet of relaxed inequality
   $\Rightarrow$ drop $S_2$ and relax adjacent inequality of $S_1$
Set Coalescing

Case distinction

1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
2. no separating constraints and cut constraints of $S_2$ are valid for cut facets of $S_1$
3. single pair of adjacent inequalities (other constraints valid)
4. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + constraints of $S_2$ valid for facet of relaxed inequality
5. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + other constraints of $S_1$ are valid
   + inequality and equality can be wrapped to include union
   $\Rightarrow$ replace $S_1$ and $S_2$ by valid and wrapping constraints

6. $S_2$ extends beyond $S_1$ by at most one and all cut constraints of $S_2$ and parallel slices of $S_1$ can be wrapped to include union
   $\Rightarrow$ replace $S_1$ and $S_2$ by valid and wrapping constraints
Set Coalescing

Case distinction

1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
2. no separating constraints and cut constraints of $S_2$ are valid for cut facets of $S_1$
3. single pair of adjacent inequalities (other constraints valid)
4. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + constraints of $S_2$ valid for facet of relaxed inequality
5. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + other constraints of $S_1$ are valid
   + inequality and equality can be wrapped to include union
   $\Rightarrow$ replace $S_1$ and $S_2$ by valid and wrapping constraints

$S_2$ extends beyond $S_1$ by at most one and all cut constraints of $S_1$ and parallel slices of $S_2$ can be wrapped to include union
$\Rightarrow$ replace $S_1$ and $S_2$ by valid and wrapping constraints
Set Coalescing

Case distinction

1. non-redundant constraints of $S_1$ are valid for $S_2$, i.e., $S_2 \subseteq S_1$
2. no separating constraints and cut constraints of $S_2$ are valid for cut facets of $S_1$
3. single pair of adjacent inequalities (other constraints valid)
4. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + constraints of $S_2$ valid for facet of relaxed inequality
5. single adjacent pair of an inequality ($S_1$) and an equality ($S_2$)
   + inequality and equality can be wrapped to include union
6. $S_2$ extends beyond $S_1$ by at most one and all cut constraints of $S_1$ and parallel slices of $S_2$
can be wrapped to include union

$\Rightarrow$ replace $S_1$ and $S_2$ by valid and wrapping constraints
Positive Powers

Definition (Power of a Relation)

Let $R$ be a Presburger relation and $k$ a positive integer, then power $k$ of relation $R$ is defined as

$$R^k := \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2 \end{cases}$$

Example

$$R = \{ [x] \rightarrow [x + 1] \}$$

$$R^k = \{ [x] \rightarrow [x + k] : k \geq 1 \}$$
Transitive Closures

Definition (Transitive Closure of a Relation)

Let $R$ be a Presburger relation, then the transitive closure $R^+$ of $R$ is the union of all positive powers of $R$,

$$R^+ := \bigcup_{k \geq 1} R^k.$$  

Example

$R = \{ [x] \rightarrow [x + 1] \}$

$R^k = \{ [x] \rightarrow [x + k] : k \geq 1 \}$

$R^+ = \{ [x] \rightarrow [y] : \exists k \geq 1 : y = x + k \} = \{ [x] \rightarrow [y] : y \geq x + 1 \}$

Definition (Transitive Closure of a Relation, Alternative)

Inductive definition:

$$R^+ := R \cup (R \circ R^+)$$
Transitive Closures — Approximation

Fact

Given a Presburger relation $R$, the power $R^k$ (with $k$ a parameter) and the transitive closure $R^+$ may not be Presburger relations.

Example

$$R = \{ [x] \rightarrow [2x] \}$$
$$R^k = \{ [x] \rightarrow [2^k x] \}$$

⇒ need for approximation
  ▶ overapproximation $R^+$
  ▶ underapproximation $R^-$

Note

Do not use transitive closures if there is an alternative.
Transitive Closures — Graph Example

Given a graph (represented as a Presburger relation)

\[ M = \{ A[i] \rightarrow A[i+1] : 0 \leq i \leq 3; B[] \rightarrow A[2] \} \]

What is the transitive closure?

\[ M^+ = \{ A[i] \rightarrow A[i'] : 0 \leq i < i' \leq 4; B[] \rightarrow A[i] : 2 \leq i \leq 4 \} \]
Conclusion

isl is a versatile tool for polyhedral compilation and beyond

Combination of

- high-level interface
- core algorithms
- heuristics

Possible future extensions

- function symbols
- n-ary relations
- other constraint types
- partially defined piecewise quasi polynomial expression
- cardinality
References I


References II


References III


References IV


References V