Polyhedral Compilation and the Integer Set Library

Sven Verdoolaege

Cerebras Systems

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

July 4, 2022

Outline

Cerebras Wafer-Scale Engine (WSE-2)

The Largest Chip in the World

850,000 cores optimized for sparse linear algebra

46,225 mm^2 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
- …

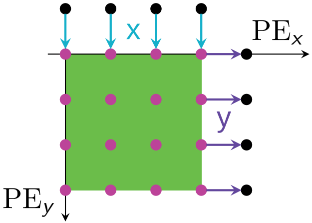
$\Rightarrow$ Cerebras `DTG` tool

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

Automatic Code Generation


```
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 32×16 matrix vector multiplication to 4×4 PEs.




```
size: { PE[4, 4] }
compute_map: { MV[i, j] -> PE[j//4, i//8] }
iport_map: { x[i=0:15] -> [PE[i//4, -1] -> index[i%4]] }
oport_map: { y[i=0:31] -> [PE[4, i//8] -> index[i%8]] }
```

Affine Constraints

Computation instances, tensor elements, PE coordinates, ordering
$\Rightarrow$ represented by a tuple of integers

- Set of computation instances $\{\,\mathrm{MV}[i,j] : 0 \le i < \mathrm{M} \wedge 0 \le j < \mathrm{N}\,\}$
  $\Rightarrow$ rectangle of fixed size
- Accesses $\{\,\mathrm{MV}[i,j] \to \mathrm{x}[j]\,\} \cup \{\,\mathrm{MV}[i,j] \to \mathrm{y}[i]\,\} \cup \{\,\mathrm{MV}[i,j] \to \mathrm{W}[i,j]\,\}$
  $\Rightarrow$ affine in instance identifiers
- Placement $\{\,\mathrm{MV}[i,j] \to \mathrm{PE}[\lfloor j/4 \rfloor, \lfloor i/8 \rfloor]\,\}$
  $\Rightarrow$ quasi affine (may involve integer divisions)
- Communication $\{\,\mathrm{x}[i = 0{:}15] \to [\mathrm{PE}[\lfloor i/4 \rfloor, -1] \to \mathrm{index}[i \bmod 4]]\,\}$
  $\Rightarrow$ quasi affine

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

$\Rightarrow$ Polyhedral Compilation

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
  - $\Rightarrow$ the set of all statement instances
- Access Relations
  - $\Rightarrow$ the array elements accessed by a statement instance
- Dependences
  - $\Rightarrow$ the statement instances that depend on a statement instance
- Schedule
  - $\Rightarrow$ the relative execution order of statement instances

Illustrative Example: Matrix Multiplication

```
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)

$$\{\, \mathtt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N; \mathtt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

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

$$W = \{\, \mathtt{S1}[i,j] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j] \,\}$$

$$R = \{\, \mathtt{S2}[i,j,k] \to \mathtt{C}[i,j]; \mathtt{S2}[i,j,k] \to \mathtt{A}[i,k]; \mathtt{S2}[i,j,k] \to \mathtt{B}[k,j] \,\}$$

- Schedule (relative execution order)

$$\{\, \mathtt{S1}[i,j] \to [i,j,0,0]; \mathtt{S2}[i,j,k] \to [i,j,1,k] \,\}$$

Presburger Sets and Relations

Examples

$$\{ \texttt{S1}[i,j] : 0 \le i < M \wedge 0 \le j < N; \texttt{S2}[i,j,k] : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \}$$

$$\{ \boxed{\texttt{S1}[i,j] \to \texttt{C}[i,j]}; \boxed{\texttt{S2}[i,j,k] \to \texttt{C}[i,j]} \}$$

General form

- Sets

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

  with f_k Presburger formulas

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

- Binary relations

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

  $\Rightarrow$ set of pairs of elements of the form $S_1[\mathbf{i}] \to T_1[\mathbf{j}]$

  Note: despite "$\to$", not necessarily (single valued) functions

Quasi-affine Expressions and Presburger Formulas

- quasi-affine expression (no multiplication; only constant functions)
  - variable x
  - constant integer number 3
  - constant symbol N
  - addition $(+)$, subtraction $(-)$ $x+3$
  - integer division by integer constant d $(\lfloor \cdot / d \rfloor)$ $\lfloor (x+3)/16 \rfloor$
- Presburger formula
  - true
  - quasi-affine expression
  - less-than-or-equal relation $(\leq)$ $0 \leq x$
  - equality $(=)$
  - first order logic connectives: $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$ $0 \leq x \wedge x < N$
- not allowed: multiplication, functions with arity greater than zero
  $x * x$, $x * N$, f(x)
- allowed: repeated addition
  $3 * x \equiv x + x + x$

Presburger Sets and Relations

General form

- Sets

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

where $f_k(\mathbf{i})$ are Presburger formulas with $\mathbf{i}$ as only free variables
$\Rightarrow$ set of elements of the form $S_1[\mathbf{i}]$, one for each $\mathbf{i}$ such that $f_1(\mathbf{i})$ is true, ...

Note: may depend on interpretation of symbolic constants

$$\{ \mathrm{S}[i] : 0 \le i \le \mathrm{n} \}$$

is equal to

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

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

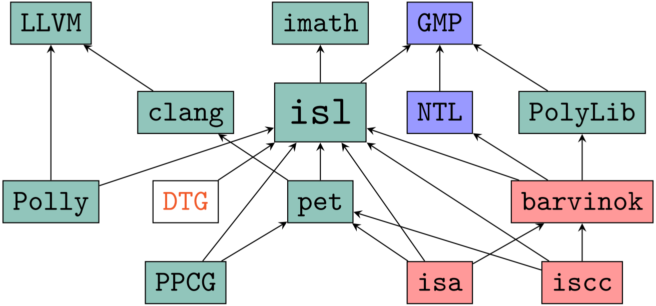
Supported operations by core library include

- *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



Licenses:
BSD/MIT/
Apache
LGPL
GPL

`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

Set Representation

[10]

```
S:  A[0] = 1;
    for (i = 1; i < N; ++i)
T:      A[i] = 2 * A[i - 1];
```

- `isl`: named (and nested) spaces

```
[N] -> { S[]; T[i]: 1 <= i < N }
```

- `Omega`:

S padding T

```
symbolic N;
{ [0, 0] } union { [1, i]: 1 <= i < N}
```

- `PolyLib`: (deals with rational sets, polyhedra)

```
2
2 5
0  1  0  0  0
0  0  1  0  0
3 5
0  1  0  0 -1
1  0  1  0 -1
1  0 -1  1 -1
```

equality/inequality N

Spaces

Recall general form

- Sets

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

- Binary relations

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

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., $\mathbf{i}$, $\mathbf{j}$)

A statement $S_2[\mathbf{i}] = T_1[\mathbf{j}]$ means

- the identifiers S_2 and T_1 are the same, and
- the sizes of $\mathbf{i}$ and $\mathbf{j}$ are the same

Examples: $\mathrm{S}[] \neq \mathrm{S}[i]$, $\mathrm{S}[a] = \mathrm{S}[b]$, $\mathrm{S}[] \neq \mathrm{T}[]$

Nested Relations

`isl` currently supports

- sets

$$\{ S_1[\mathbf{i}] : f_1(\mathbf{i}); S_2[\mathbf{i}] : f_2(\mathbf{i}); \dots \},$$

- binary relations

$$\{ S_1[\mathbf{i}] \to T_1[\mathbf{j}] : f_1(\mathbf{i}, \mathbf{j}); S_2[\mathbf{i}] \to T_2[\mathbf{j}] : f_2(\mathbf{i}, \mathbf{j}); \dots \}$$

but not

- n-ary relations

$$\{ A[\mathbf{i}] \to B[\mathbf{j}] \to C[\mathbf{k}] \to \dots \}$$

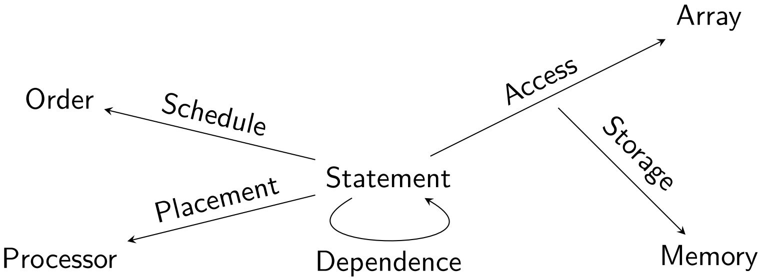
However, nested relations are supported

For example: statement instance specific memory map

$$\{ [\mathrm{S}[i, j] \to \mathrm{A}[i]] \to \mathrm{Mem}[i, j] \}$$

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

Polyhedral Objects



+ 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`)

```
{ S2[i, j, k] : 0 <= i < M and 0 <= j < N and 0 <= k < K }
{ S1[i, j] -> C[i, j] }
```

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

```
{ S1[i, j] -> C[i, j]; S2[i, j, k] -> 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

[17]

- 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,

```
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)

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

Each disjunct consists of

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

Conversion to disjunction of conjunctions

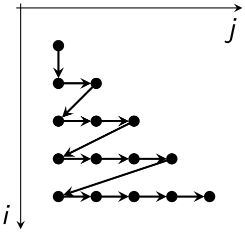
$$\neg(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a})) \quad \to \quad \neg f(\mathbf{x}, g(\mathbf{x}))$$

$\Rightarrow$ determine a single value of $\mathbf{a}$ satisfying $f(\mathbf{x}, \mathbf{a})$ and write it as an explicit piecewise quasi affine expression $g(\mathbf{x})$ of $\mathbf{x}$

$\Rightarrow$ 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:

$$\mathbf{a} \prec \mathbf{b} \equiv \bigvee_{i=1}^{n} \left(a_i < b_i \wedge \bigwedge_{j=1}^{i-1} a_j = b_j \right)$$

$\Rightarrow$ smaller in first position where tuples differ

Parametric Integer Programming

[3]

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

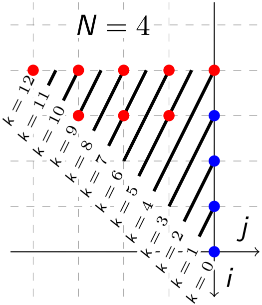
Result:

- 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

$$R = \{\, [i,j] : 0 \le -i \le N \wedge 0 \le -j \le -i \wedge 0 \le k \le 3N \wedge k = -i - 2j \,\}$$



lexmin $R =$

if $k < N$

$[-k, 0]$

else

if $3\left\lfloor \frac{k+N}{2} \right\rfloor \ge 2k$

$\left[k - 2\left\lfloor \frac{k+N}{2} \right\rfloor, -k + \left\lfloor \frac{k+N}{2} \right\rfloor\right]$

Parametric Integer Programming on Presburger Sets and Relations

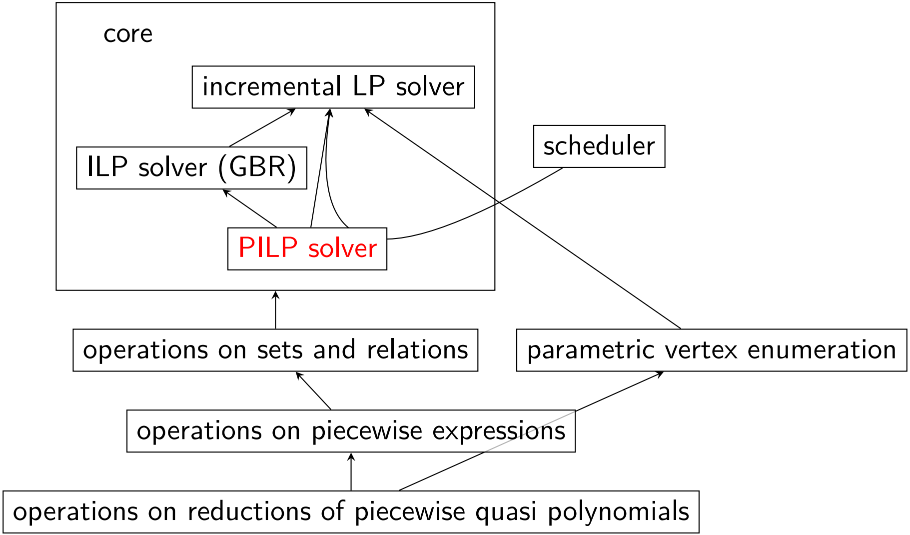
$$R = \bigcup_i R_i \qquad R_i = \{\, S[\mathbf{i}] \to T[\mathbf{j}] : \exists \mathbf{k} : A_0\mathbf{c} + A_1\mathbf{i} + A_2\mathbf{j} + A_3\mathbf{k} \geq \mathbf{a} \,\}$$

- Compute $\operatorname{lexmin} R$
  $\Rightarrow$ treat R_i as a parametric polyhedron with
  - parameters $\mathbf{c}$ and $\mathbf{i}$
  - variables $\mathbf{j}$ and $\mathbf{k}$

  $\Rightarrow$ combine results over multiple disjuncts
- Quantifier elimination
  $\Rightarrow$ treat R_i as a parametric polyhedron with
  - parameters $\mathbf{c}$, $\mathbf{i}$ and $\mathbf{j}$
  - variables $\mathbf{k}$

Internal Structure of isl

[1, 2, 3, 8, 9, 13]



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
  $\{ [i] \to [j, k] : i - 3 \leq 4j \leq i \wedge j \leq k \leq j + 1 \}$
  - 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 $\sum_i a_i x_i \leq f_j(\mathbf{n})$
  $\Rightarrow$ replace by $\sum_i a_i x_i \leq u$ with $u \leq f_j(\mathbf{n})$ extra parameter
  $\Rightarrow$ avoid considering all orderings of $f_j(\mathbf{n})$
- combine cells with same expression for minimum

[7]

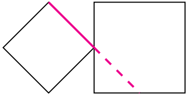
Choice of Internal Representation

Quantifier elimination

- `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

- `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

$$f(\mathbf{i}) = 0 \qquad \Rightarrow \qquad f(\mathbf{i}) \geq 0 \wedge f(\mathbf{i}) \leq 0$$

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

Other "redundant" types of constraints could also be useful

- disequality constraint

$$f(\mathbf{i}) \neq 0 \qquad \Leftarrow \qquad f(\mathbf{i}) \geq 1 \vee f(\mathbf{i}) \leq -1$$

- lexicographic constraint

$$\mathbf{a} \prec \mathbf{b} \qquad \Leftarrow \qquad \bigvee_{i=1}^{n} \left(a_i < b_i \wedge \bigwedge_{j=1}^{i-1} a_j = b_j \right)$$

$\Rightarrow$ adjust core algorithms or expand before applying

Piecewise Expressions

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, symbolic constants, integer constants, addition ($+$), subtraction ($-$) and integer division by a constant ($\lfloor \cdot / d \rfloor$)
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, symbolic constants, rational constants, addition ($+$), subtraction ($-$) and multiplication ($\cdot$)
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
  - $\Rightarrow$ a list of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

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

Piecewise Expressions

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

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

- Piecewise quasi affine expression *typically* represents element of set (e.g., 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 `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

$$R = \{\, S[\mathbf{i}] \to S[\mathbf{j}] : P(\mathbf{i}, \mathbf{j}) \,\}$$

$$\Delta R = \{\, S[\mathbf{k}] : \exists \mathbf{i}, \mathbf{j} : S[\mathbf{i}] \to S[\mathbf{j}] \in R \wedge \mathbf{k} = \mathbf{j} - \mathbf{i} \,\}$$

Example:

$$R = \{\, \mathrm{S}[i_1, i_2] \to \mathrm{S}[0, j_2] : 0 \le i_1 \le 10 \wedge 0 \le i_2 \le 10 \wedge i_2 \le j_2 \le i_2 + 2 \,\}$$

$$\Delta R = \{\, \mathrm{S}[k_1, k_2] : -10 \le k_1 \le 0 \wedge 0 \le k_2 \le 2 \,\}$$

- Elements of ΔR live in same space as domain and range of R
  Does it make sense to intersect ΔR with $\operatorname{dom} R$?
- 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

$$\{\, \mathrm{S1}[i_1, i_2] \to \mathrm{S2}[j_1, j_2] : \ldots \,\}$$

Set Coalescing

[11]

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

$$S_1 = \{\, \mathbf{x} : A\mathbf{x} \ge \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} : B\mathbf{x} \ge \mathbf{d} \,\}$$

`PolyLib` way:

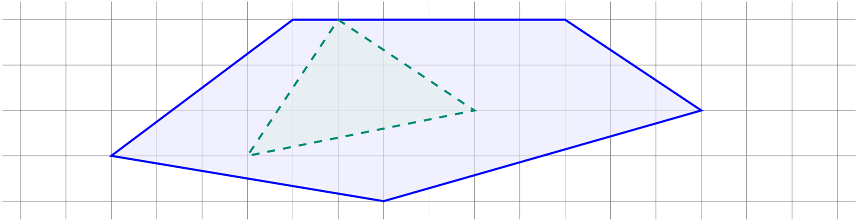
1. Compute $H = \operatorname{conv.hull}(S_1 \cup S_2)$
2. Replace $S_1 \cup S_2$ by $H \setminus (H \setminus (S_1 \cup S_2))$

`isl` way:

1. Classify constraints
   - redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints of S_1
   - valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
   - separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2; special cases:
     - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
     - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
   - cut: otherwise

Set Coalescing

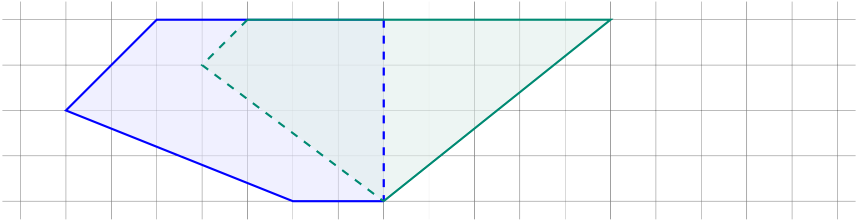
[11]



2. 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

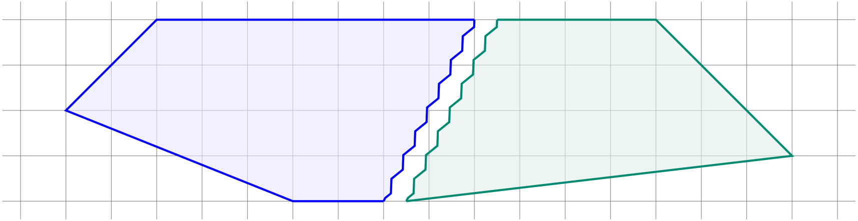
[11]



2. 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

[11]

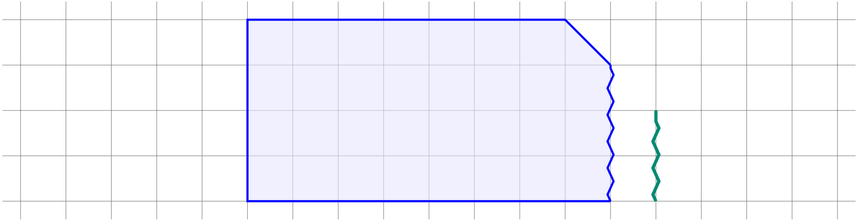


2. Case distinction
   1. non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
   2. no separating constraints and cut constraints of S_2 are valid for cut facets of S_1
   3. single pair of adjacent inequalities (other constraints valid)

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

Set Coalescing

[11]



2. 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)

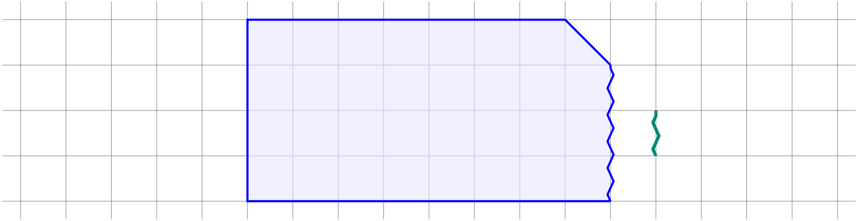
      $+$ other constraints of S_1 are valid

      $+$ constraints of S_2 valid for facet of relaxed inequality

      $\Rightarrow$ drop S_2 and relax adjacent inequality of S_1

Set Coalescing

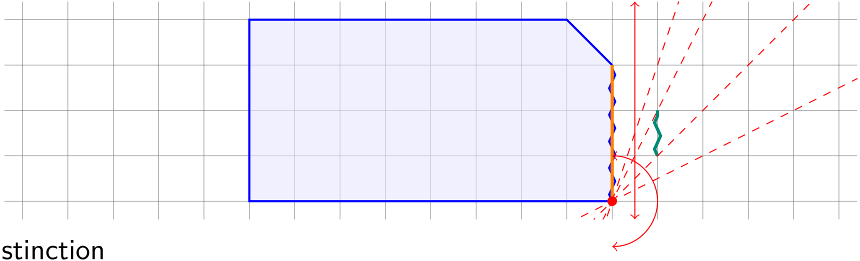
[11]



2. 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

Set Coalescing

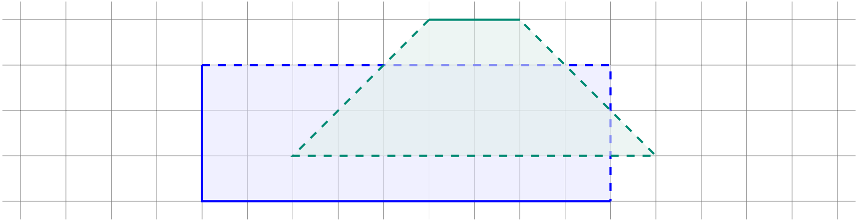
[11]



2. 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

Set Coalescing

[11]



2. 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

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

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

$$\begin{aligned} R &= \{ [x] \to [x+1] \} \\ R^k &= \{ [x] \to [x+k] : k \geq 1 \} \\ R^+ &= \{ [x] \to [y] : \exists k \geq 1 : y = x + k \} = \{ [x] \to [y] : y \geq x + 1 \} \end{aligned}$$

Definition (Transitive Closure of a Relation, Alternative)

Inductive definition:

$$R^+ := R \cup \left(R \circ R^+ \right)$$

[6]

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

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

$\Rightarrow$ need for approximation

- overapproximation $R^{\overline{+}}$
- underapproximation $R^{\underline{+}}$

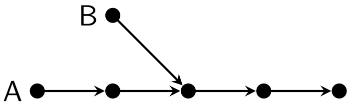
Note

Do not use transitive closures if there is an alternative.

Transitive Closures — Graph Example

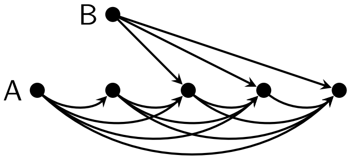
Given a graph (represented as a Presburger relation)

$$M = \{\, \mathrm{A}[i] \to \mathrm{A}[i+1] : 0 \le i \le 3; \mathrm{B}[] \to \mathrm{A}[2] \,\}$$



What is the transitive closure?

$\Rightarrow$ $M^+ = \{\, \mathrm{A}[i] \to \mathrm{A}[i'] : 0 \le i < i' \le 4; \mathrm{B}[] \to \mathrm{A}[i] : 2 \le i \le 4 \,\}$



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

[1] William Cook, Thomas Rutherford, Herbert E. Scarf, and David F. Shallcross. *An Implementation of the Generalized Basis Reduction Algorithm for Integer Programming*. Cowles Foundation Discussion Papers 990. Cowles Foundation, Yale University, Aug. 1991.

[2] David Detlefs, Greg Nelson, and James B. Saxe. "Simplify: a theorem prover for program checking". In: *J. ACM* 52.3 (2005), pp. 365–473. DOI: 10.1145/1066100.1066102.

[3] Paul Feautrier. "Parametric Integer Programming". In: *RAIRO Recherche Opérationnelle* 22.3 (1988), pp. 243–268.

[4] Tobias Grosser, Armin Größlinger, and Christian Lengauer. "Polly - Performing polyhedral optimizations on a low-level intermediate representation". In: *Parallel Processing Letters* 22.04 (2012). DOI: 10.1142/S0129626412500107.

[5] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and David Wonnacott. *The Omega Library*. Tech. rep. University of Maryland, Nov. 1996.

References II

[6] Wayne Kelly, William Pugh, Evan Rosser, and Tatiana Shpeisman. "Transitive closure of infinite graphs and its applications". In: *International Journal of Parallel Programming* 24.6 (1996), pp. 579–598. DOI: 10.1007/BFb0014196.

[7] Shubhang Kulkarni and Michael Kruse. "Polyhedral Binary Decision Diagrams for Representing Non-Convex Polyhedra". In: *12th International Workshop on Polyhedral Compilation Techniques (IMPACT'22)*. Budapest, Hungary, June 2022.

[8] Vincent Loechner and Doran K. Wilde. "Parameterized Polyhedra and Their Vertices". In: *International Journal of Parallel Programming* 25.6 (Dec. 1997), pp. 525–549. DOI: 10.1023/A:1025117523902.

[9] Sven V. "isl: An Integer Set Library for the Polyhedral Model". In: *Mathematical Software - ICMS 2010*. Ed. by Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama. Vol. 6327. Lecture Notes in Computer Science. Springer, 2010, pp. 299–302. DOI: 10.1007/978-3-642-15582-6_49.

References III

[10] Sven V. "Counting Affine Calculator and Applications". In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*. Chamonix, France, Apr. 2011. DOI: 10.13140/RG.2.1.2959.5601.

[11] Sven V. "Integer Set Coalescing". In: *Proceedings of the 5th International Workshop on Polyhedral Compilation Techniques*. Amsterdam, The Netherlands, Jan. 2015. DOI: 10.13140/2.1.1313.6968.

[12] Sven V. and Tobias Grosser. "Polyhedral Extraction Tool". In: *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*. Paris, France, Jan. 2012. DOI: 10.13140/RG.2.1.4213.4562.

[13] Sven V. and Gerda Janssens. *Scheduling for PPCG*. Report CW 706. Leuven, Belgium: Department of Computer Science, KU Leuven, June 2017. DOI: 10.13140/RG.2.2.28998.68169.

References IV

[14] Sven V., Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. "Polyhedral parallel code generation for CUDA". In: *ACM Trans. Archit. Code Optim.* 9.4 (2013), p. 54. DOI: 10.1145/2400682.2400713.

[15] Sven V., Manjunath Kudlur, Rob Schreiber, and Harinath Kamepalli. "Generating SIMD Instructions for Cerebras CS-1 using Polyhedral Compilation Techniques". In: *10th International Workshop on Polyhedral Compilation Techniques (IMPACT'20)*. Bologna, Italy, Jan. 2020. DOI: 10.5281/zenodo.4295955.

[16] Sven V., Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. "Counting integer points in parametric polytopes using Barvinok's rational functions". In: *Algorithmica* 48.1 (June 2007), pp. 37–66. DOI: 10.1007/s00453-006-1231-0.

[17] Sven V., Oleksandr Zinenko, Manjunath Kudlur, Ron Estrin, Tianjiao Sun, and Harinath Kamepalli. "A Templated C++ Interface for isl". In: *11th International Workshop on Polyhedral Compilation Techniques (IMPACT'21)*. Jan. 2021. DOI: 10.5281/zenodo.6670306.

References V

[18] Doran K. Wilde. *A Library for doing polyhedral operations*. Tech. rep. 785. IRISA, Rennes, France, 1993, 45 p.