Recent Advances in Algorithms Supporting the Polyhedral Model

Marc Moreno Maza [1]
[1]Western University

CDP 2024, November 12, 2024

Polyhedral model (1/2)

- The **polyhedral model** is a mathematical description for representing and manipulating static control parts (SCoPs) of a program by using Presburger's arithmetic.

Polyhedral model (1/2)

- The **polyhedral model** is a mathematical description for representing and manipulating static control parts (SCoPs) of a program by using Presburger's arithmetic.
- A **static control part** (SCoP) is a maximal set of consecutive statements without while loops, where loop bounds and conditionals may only depend on invariants within this set of statements.

Polyhedral model (1/2)

- The **polyhedral model** is a mathematical description for representing and manipulating static control parts (SCoPs) of a program by using Presburger's arithmetic.
- A **static control part** (SCoP) is a maximal set of consecutive statements without while loops, where loop bounds and conditionals may only depend on invariants within this set of statements.
- Various libraries such as **ISL, Polylib, Piplib, Omegalib** implement the polyhedral model's underlying mathematical op- erations.

Polyhedral model (1/2)

- The **polyhedral model** is a mathematical description for representing and manipulating static control parts (SCoPs) of a program by using Presburger's arithmetic.
- A **static control part** (SCoP) is a maximal set of consecutive statements without while loops, where loop bounds and conditionals may only depend on invariants within this set of statements.
- Various libraries such as **ISL, Polylib, Piplib, Omegalib** implement the polyhedral model's underlying mathematical op- erations.
- The model provides an abstract representation of for-loop nests that enables us to optimize them via different transformations, including: **loop blocking (tiling), loop parallelizing, …**

Polyhedral model (2/2)

The model is based on **statement instances** in a for-loop nest.
It uses four main components to represent the abstraction of a program:

1- **iteration domain**: integer polyhedron of all iteration instances
2- **access relations**: access relations of iteration instances
3- **schedule**: the order of execution
4- **dependency relations**: read/write dependencies.

```
for(i=0; i<2*N+5; i++){
  for(j=0; j<i; j++)
 S: A[i][j] = A[i][j-1]*2;
 /* statement instances:
  {<S,[0,0]>, <S,[0,1]>, ...}
  iteration domain:
  {0 <= i < 2*N+5, 0 <= j < i
   }
  access relations:
  {[i;j], [i;j-1]}
  schedule: lexicographical
  dependency:
  <S,[i,j]> -> <S,[i,j-1]> */

}
```

Overview of the talk

1. Efficient detection of redundancies in systems of linear inequalities (ISSAC 2024) . Joint work with Rui-Juan Jing (Jiangsu University), Yan-Feng Xie (Chinese Academy of Sciences, Beijing) and Chun-Ming Yuan (Chinese Academy of Sciences, Beijing).

Overview of the talk

1. Efficient detection of redundancies in systems of linear inequalities (ISSAC 2024) . Joint work with Rui-Juan Jing (Jiangsu University), Yan-Feng Xie (Chinese Academy of Sciences, Beijing) and Chun-Ming Yuan (Chinese Academy of Sciences, Beijing).
2. Computing the Integer Hull of Convex Polyhedral Sets (CASC 2022) . Joint work with Lin-Xiao Wang (Microsoft).

Overview of the talk

1. Efficient detection of redundancies in systems of linear inequalities (ISSAC 2024) . Joint work with Rui-Juan Jing (Jiangsu University), Yan-Feng Xie (Chinese Academy of Sciences, Beijing) and Chun-Ming Yuan (Chinese Academy of Sciences, Beijing).
2. Computing the Integer Hull of Convex Polyhedral Sets (CASC 2022) . Joint work with Lin-Xiao Wang (Microsoft).
3. A Pipeline Pattern Detection Technique in Polly (IMPACT 22, LLPP 22) . Joint work with Delaram Talaashrafi (NVIADIA) and Johannes Doerfert (Argonne National Laboratory).

Outline

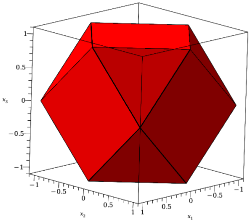
Efficient detection of redundancies in systems of linear inequalities

Faster computations of integer hulls fo polyhedral sets

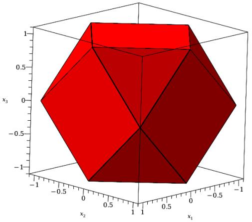
A Pipeline Pattern Detection Technique in Polly

$$
\begin{cases}
-x_3 \leq 1 \\
-x_1 - x_2 - x_3 \leq 2 \\
-x_1 + x_2 - x_3 \leq 2 \\
x_1 - x_2 - x_3 \leq 2 \\
x_1 + x_2 - x_3 \leq 2 \\
x_3 0 \leq 1 \\
-x_1 - x_2 + x_3 \leq 2 \\
-x_1 + x_2 + x_3 \leq 2 \\
x_1 - x_2 + x_3 \leq 2 \\
x_1 + x_2 + x_3 \leq 2 \\
-x_2 0 \leq 1 \\
x_2 \leq 1 \\
-x_1 \leq 1 \\
x_1 0 \leq 1
\end{cases}
$$

$$
\begin{cases}
-x_3 \leq 1 \\
-x_1 - x_2 - x_3 \leq 2 \\
-x_1 + x_2 - x_3 \leq 2 \\
x_1 - x_2 - x_3 \leq 2 \\
x_1 + x_2 - x_3 \leq 2 \\
x_3 0 \leq 1 \\
-x_1 - x_2 + x_3 \leq 2 \\
-x_1 + x_2 + x_3 \leq 2 \\
x_1 - x_2 + x_3 \leq 2 \\
x_1 + x_2 + x_3 \leq 2 \\
-x_2 0 \leq 1 \\
x_2 \leq 1 \\
-x_1 \leq 1 \\
x_1 0 \leq 1
\end{cases}
$$

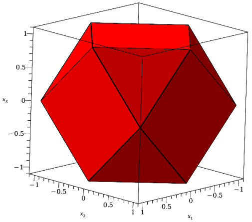


$$
\begin{cases}
-x_3 \leq 1 \\
-x_1 - x_2 - x_3 \leq 2 \\
-x_1 + x_2 - x_3 \leq 2 \\
x_1 - x_2 - x_3 \leq 2 \\
x_1 + x_2 - x_3 \leq 2 \\
x_3 0 \leq 1 \\
-x_1 - x_2 + x_3 \leq 2 \\
-x_1 + x_2 + x_3 \leq 2 \\
x_1 - x_2 + x_3 \leq 2 \\
x_1 + x_2 + x_3 \leq 2 \\
-x_2 0 \leq 1 \\
x_2 \leq 1 \\
-x_1 \leq 1 \\
x_1 0 \leq 1
\end{cases}
$$

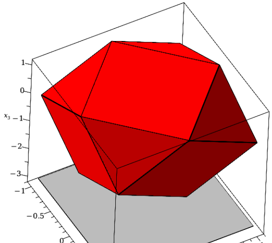


$$
\begin{cases}
0 \leq 1 + x_2 \\
0 \leq 1 - x_2 \\
0 \leq x_1 + 1 \\
0 \leq 1 - x_1
\end{cases}
$$

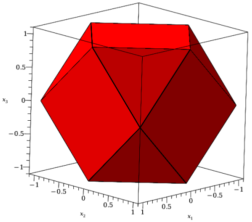
$$
\begin{cases}
-x_3 \leq 1 \\
-x_1 - x_2 - x_3 \leq 2 \\
-x_1 + x_2 - x_3 \leq 2 \\
x_1 - x_2 - x_3 \leq 2 \\
x_1 + x_2 - x_3 \leq 2 \\
x_3 0 \leq 1 \\
-x_1 - x_2 + x_3 \leq 2 \\
-x_1 + x_2 + x_3 \leq 2 \\
x_1 - x_2 + x_3 \leq 2 \\
x_1 + x_2 + x_3 \leq 2 \\
-x_2 0 \leq 1 \\
x_2 \leq 1 \\
-x_1 \leq 1 \\
x_1 0 \leq 1
\end{cases}
$$



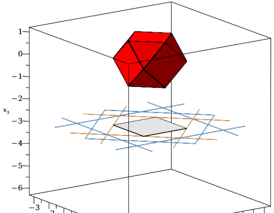
$$
\begin{cases}
0 \leq 1 + x_2 \\
0 \leq 1 - x_2 \\
0 \leq x_1 + 1 \\
0 \leq 1 - x_1
\end{cases}
$$



$$\begin{cases} -x_3 \le 1 \\ -x_1 - x_2 - x_3 \le 2 \\ -x_1 + x_2 - x_3 \le 2 \\ x_1 - x_2 - x_3 \le 2 \\ x_1 + x_2 - x_3 \le 2 \\ x_3 0 \le 1 \\ -x_1 - x_2 + x_3 \le 2 \\ -x_1 + x_2 + x_3 \le 2 \\ x_1 - x_2 + x_3 \le 2 \\ x_1 + x_2 + x_3 \le 2 \\ -x_2 0 \le 1 \\ x_2 \le 1 \\ -x_1 \le 1 \\ x_1 0 \le 1 \end{cases}$$



$$\begin{cases} 0 \le 1 + x_2 \\ 0 \le 1 - x_2 \\ 0 \le x_1 + 1 \\ 0 \le 1 - x_1 \end{cases}$$



Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

skip slide

Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

Dependence analysis yields:

$$(t, p) := (n - j, i + j).$$

▸▸ skip slide

Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

Dependence analysis yields:
$(t, p) := (n - j, i + j)$.

$$\begin{cases} 0 \leq & i \\ i \leq & n \\ 0 \leq & j \\ j \leq & n \\ t = & n - j \\ p = & i + j \end{cases}$$

skip slide

Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

Dependence analysis yields:
$(t, p) := (n - j, i + j)$.

$$\begin{cases} 0 \leq & i \\ i \leq & n \\ 0 \leq & j \\ j \leq & n \\ t = & n - j \\ p = & i + j \end{cases}$$

FME reorders $p > t > i > j > n$ to $i > j > t > p > n$, thus eliminating i, j. ⏩ skip slide

Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

Dependence analysis yields:
$(t, p) := (n - j, i + j)$.

$$\begin{cases} 0 \leq & i \\ i \leq & n \\ 0 \leq & j \\ j \leq & n \\ t = & n - j \\ p = & i + j \end{cases} \qquad \begin{cases} i = & p + t - n \\ j = & -t + n \\ t \geq & \max(0, -p + n) \\ t \leq & \min(n, -p + 2n) \\ 0 \leq & p \\ p \leq & 2n \\ 0 \leq & n. \end{cases}$$

FME reorders $p > t > i > j > n$ to $i > j > t > p > n$, thus eliminating i, j. skip slide

Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

Dependence analysis yields: $(t, p) := (n - j, i + j)$.

$$\begin{cases} 0 \leq & i \\ i \leq & n \\ 0 \leq & j \\ j \leq & n \\ t = & n - j \\ p = & i + j \end{cases}$$

The new representation allows us to generate the multithreaded code.

$$\begin{cases} i = & p + t - n \\ j = & -t + n \\ t \geq & \max(0, -p + n) \\ t \leq & \min(n, -p + 2n) \\ 0 \leq & p \\ p \leq & 2n \\ 0 \leq & n. \end{cases}$$

FME reorders $p > t > i > j > n$ to $i > j > t > p > n$, thus eliminating i, j. skip slide

Application of FME: code generation

```
for(i=0; i<=n; i++){
  c[i] = 0; c[i+n] = 0;
  for(j=0; j<=n; j++)
    c[i+j] += a[i]*b[j];
}
```

```
parallel_for (p=0; p<=2*n; p++){
  c[p] = 0;
  for (t=max(0,n-p);
      t<=min(n,2*n-p);t++)
    c[p] += A[t+p-n] * B[n-t];
}
```

Dependence analysis yields: $(t,p) := (n-j, i+j)$.

$$\begin{cases} 0 \le & i \\ i \le & n \\ 0 \le & j \\ j \le & n \\ t = & n-j \\ p = & i+j \end{cases}$$

The new representation allows us to generate the multithreaded code.

$$\begin{cases} i = & p+t-n \\ j = & -t+n \\ t \ge & \max(0, -p+n) \\ t \le & \min(n, -p+2n) \\ 0 \le & p \\ p \le & 2n \\ 0 \le & n. \end{cases}$$

FME reorders $p > t > i > j > n$ to $i > j > t > p > n$, thus eliminating i, j. skip slide

Removing redundant inequalities in FME

- Removing redundant inequalities, when solving a linear inequality system, is crucial for improving computational efficiency, for numerical stability, and for the interpretability of the result.

Removing redundant inequalities in FME

- Removing redundant inequalities, when solving a linear inequality system, is crucial for improving computational efficiency, for numerical stability, and for the interpretability of the result.
- Linear programming (LP) has been widely used for this task. .

Removing redundant inequalities in FME

- Removing redundant inequalities, when solving a linear inequality system, is crucial for improving computational efficiency, for numerical stability, and for the interpretability of the result.
- Linear programming (LP) has been widely used for this task. .
- Other approaches take advantage of duality in the theory of polyhedral. Then, the redundant inequalities can be detected by checking the ranks of specific matrices over $\mathbb{Q}$.

Removing redundant inequalities in FME

- Removing redundant inequalities, when solving a linear inequality system, is crucial for improving computational efficiency, for numerical stability, and for the interpretability of the result.
- Linear programming (LP) has been widely used for this task. .
- Other approaches take advantage of duality in the theory of polyhedral. Then, the redundant inequalities can be detected by checking the ranks of specific matrices over $\mathbb{Q}$.
- In our recent paper, we further simplify the redundant detection by manipulating Boolean matrices on which we perform bit-vector arithmetic.

test case	(n, m, k)	mpr	BPAS	cdd	polylib
32hedron	(6, 32, 11)	6.54	16.80	4183.08	**1.92**
64hedron	(7,64,13)	13.05	52.42	>5min	**1.67**
francois	(13,27,2304)	499.92	**253.66**	388.36	> 5min
francois2	(13,31,384)	**41.80**	140.34	55.17	80.63
herve.in	(14,25,262)	34.42	140.34	294.01	**30.08**
c6.in	(11,17,31)	**9.85**	12.72	84.11	5.56
c9.in	(16,18,140)	**25.08**	65.54	151.17	131.53
c10.in	(18,20,142)	22.10	98.68	249.02	**16.06**
S24	(24, 25,25)	23.50	58.80	748.67	**17.47**
S35	(35, 36,36)	46.55	182.14	3575.00	**46.007**
cube	(10, 20,1024)	**81.33**	201.92	125.900	161.06
C56	(5, 6,6)	3.67	4.09	11.81	**0.79**
C1011	(10, 11,11)	24.99	115.68	1716.25	**9.99**
C510	(5, 42,10)	12.00	40.01	>5min	**4.42**
T1	(5, 10,38)	**5.61**	16.44	27.42	8.81
T3	(10,12,29)	21.29	141.64	288.07	**12.07**
T5	(5, 10,36)	8.12	15.62	22.92	**4.76**
T6	(10,20,390)	**1142.9**	23800.11	14937.61	>5min
T7	(5, 8,26)	5.81	10.79	13.96	**4.00**
T9	(10,12,36)	**36.56**	414.53	479.18	100.34
T10	(6, 8,24)	**4.58**	13.65	18.39	5.27
T12	(5, 11,42)	**8.52**	19.03	38.65	8.60
R_15_20	(15, 20,1328)	**28430.40**	336035.00	38037.21	>5min

Outline

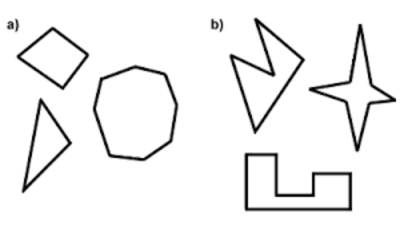
Efficient detection of redundancies in systems of linear inequalities

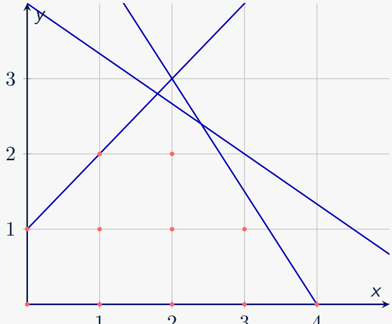
Faster computations of integer hulls fo polyhedral sets

A Pipeline Pattern Detection Technique in Polly

Convex polyhedral sets

- A subset $P \subseteq \mathbb{Q}^n$ is called a *convex polyhedral set* (or simply a *polyhedral set*) if
$$P = \{\mathbf{x} \mid A\mathbf{x} \leq \vec{b}\}$$
holds, for a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $\vec{b} \in \mathbb{Q}^m$, where n, m are positive integers.
- We are interested in computing P_I the *integer hull* of P that is the smallest convex polyhedral set containing all the integer points of P.

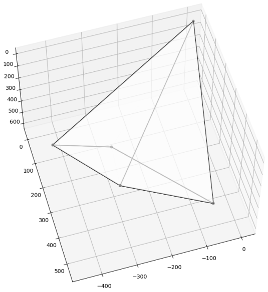




Computing integer hulls (1/3)

The input polyhedral set:

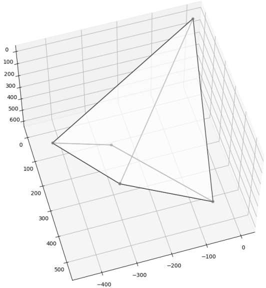
$$\left\{\begin{array}{rcl} -98877x_1 - 189663x_2 - 1798x_3 & \leq & 705915 \\ -10109x_1 - 5958x_2 - 14601x_3 & \leq & 31333 \\ -5405x_1 + 4965x_2 + 3870x_3 & \leq & 4303504 \\ 729x_1 - 117x_2 + 350x_3 & \leq & 4561 \\ 677x_1 + 465x_2 - 540x_3 & \leq & 3489 \end{array}\right.$$



Computing integer hulls (1/3)

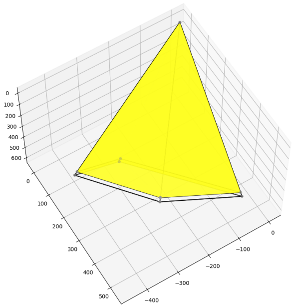
The input polyhedral set:

$$\begin{cases} -98877x_1 - 189663x_2 - 1798x_3 & \leq & 705915 \\ -10109x_1 - 5958x_2 - 14601x_3 & \leq & 31333 \\ -5405x_1 + 4965x_2 + 3870x_3 & \leq & 4303504 \\ 729x_1 - 117x_2 + 350x_3 & \leq & 4561 \\ 677x_1 + 465x_2 - 540x_3 & \leq & 3489 \end{cases}$$

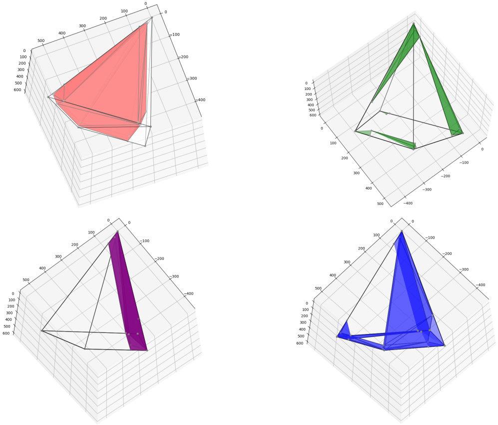


Normalization (leaves the integer hull unchanged):

$$\begin{cases} -98877x_1 - 189663x_2 - 1798x_3 & \leq & 705915 \\ -10109x_1 - 5958x_2 - 14601x_3 & \leq & 31333 \\ -1081x_1 + 993x_2 + 774x_3 & \leq & 860700 \\ 729x_1 - 117x_2 + 350x_3 & \leq & 4561 \\ 677x_1 + 465x_2 - 540x_3 & \leq & 3489 \end{cases}$$



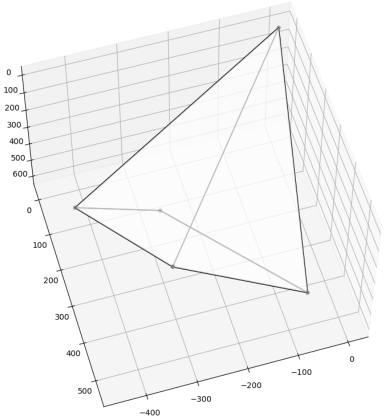
Computing integer hulls (2/3)



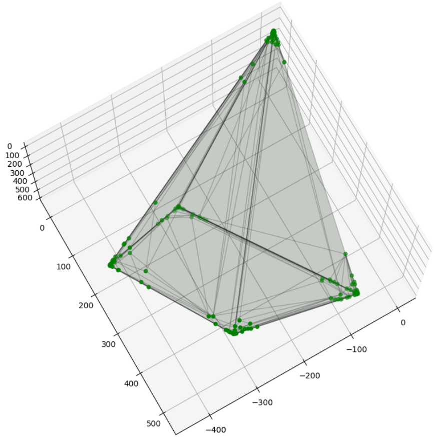
1. The **red** is an approximation of the integer hull of the input.
2. The integer hulls of border regions (green, blue, purple) are brute-force computed via FME.
3. Then `QuickHull` is applied to obtain the integer hull of the input.

Computing integer hulls (3/3)

The input has only 5 vertices.



Its integer hull has 139 vertices.



All details are in https://ir.lib.uwo.ca/etd/8985/ and in https://doi.org/10.1007/978-3-031-14788-3_14

Outline

Efficient detection of redundancies in systems of linear inequalities

Faster computations of integer hulls fo polyhedral sets

A Pipeline Pattern Detection Technique in Polly

Overview

- The polyhedral model is effective for optimizing loop nests using different methods (loop tiling, loop parallelizing, …)
- They all optimize for-loop nests on a **per-loop** basis.

Overview

- The polyhedral model is effective for optimizing loop nests using different methods (loop tiling, loop parallelizing, …)
- They all optimize for-loop nests on a **per-loop** basis.

- This work is about exploiting **cross-loop** parallelization, through tasking.
- It is done by detecting pipeline pattern between **iteration blocks of different loop nests.**
- As of 2022, we were not aware of any **fully-automatic, LLVM-based method** for detecting and exploiting parallelization opportunities between iterations of different for-loop nests through tasking.

Overview

- The polyhedral model is effective for optimizing loop nests using different methods (loop tiling, loop parallelizing, …)
- They all optimize for-loop nests on a **per-loop** basis.

- This work is about exploiting **cross-loop** parallelization, through tasking.
- It is done by detecting pipeline pattern between **iteration blocks of different loop nests.**
- As of 2022, we were not aware of any **fully-automatic, LLVM-based method** for detecting and exploiting parallelization opportunities between iterations of different for-loop nests through tasking.

We use **Polly**, an LLVM-based framework, which applies polyhedral transformations: analysis, transformation, scheduling, AST generation, code generation.

Overview

- The polyhedral model is effective for optimizing loop nests using different methods (loop tiling, loop parallelizing, ...)
- They all optimize for-loop nests on a **per-loop** basis.

- This work is about exploiting **cross-loop** parallelization, through tasking.
- It is done by detecting pipeline pattern between **iteration blocks of different loop nests.**
- As of 2022, we were not aware of any **fully-automatic, LLVM-based method** for detecting and exploiting parallelization opportunities between iterations of different for-loop nests through tasking.

We use **Polly**, an LLVM-based framework, which applies polyhedral transformations: analysis, transformation, scheduling, AST generation, code generation.

We use OpenMP, which supports **task parallelization** via:

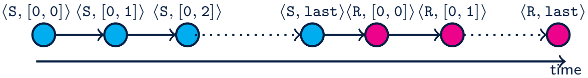
- `task` construct and `depend` clauses.

Example

```
for(i=0; i<N-1; i++)
  for(j=0; j<N-1; j++)
  S:  A[i][j]=f(A[i][j], A[i][j+1], A[
      i+1][j+1]);

for(i=0; i<N/2-1; i++)
  for(j=0; j<N/2-1; j++)
  R:  B[i][j]=g(A[i][2*j], B[i][j+1],
      B[i+1][j+1], B[i][j]);
```

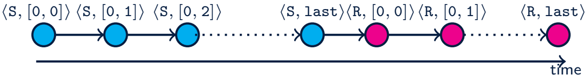
Example

```
for(i=0; i<N-1; i++)
  for(j=0; j<N-1; j++)
  S:  A[i][j]=f(A[i][j], A[i][j+1], A[i+1][j+1]);

for(i=0; i<N/2-1; i++)
  for(j=0; j<N/2-1; j++)
  R:  B[i][j]=g(A[i][2*j], B[i][j+1], B[i+1][j+1], B[i][j]);
```



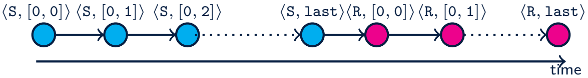
Example

```
for(i=0; i<N-1; i++)
  for(j=0; j<N-1; j++)
  S:  A[i][j]=f(A[i][j], A[i][j+1], A[
      i+1][j+1]);

for(i=0; i<N/2-1; i++)
  for(j=0; j<N/2-1; j++)
  R:  B[i][j]=g(A[i][2*j], B[i][j+1],
      B[i+1][j+1], B[i][j]);
```



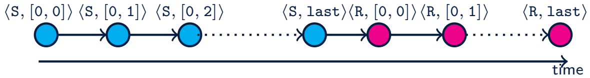
Example

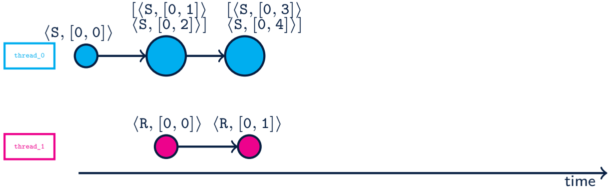
```
for(i=0; i<N-1; i++)
  for(j=0; j<N-1; j++)
  S:  A[i][j]=f(A[i][j], A[i][j+1], A[
      i+1][j+1]);

for(i=0; i<N/2-1; i++)
  for(j=0; j<N/2-1; j++)
  R:  B[i][j]=g(A[i][2*j], B[i][j+1],
      B[i+1][j+1], B[i][j]);
```



Example

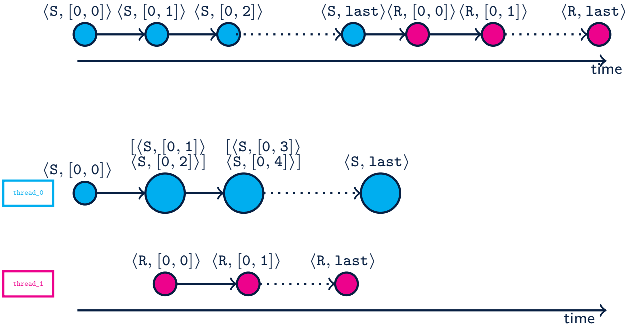
```
for(i=0; i<N-1; i++)
  for(j=0; j<N-1; j++)
  S:  A[i][j]=f(A[i][j], A[i][j+1], A[i+1][j+1]);

for(i=0; i<N/2-1; i++)
  for(j=0; j<N/2-1; j++)
  R:  B[i][j]=g(A[i][2*j], B[i][j+1], B[i+1][j+1], B[i][j]);
```





Example

```
for(i=0; i<N-1; i++)
  for(j=0; j<N-1; j++)
  S:  A[i][j]=f(A[i][j], A[i][j+1], A[
       i+1][j+1]);

for(i=0; i<N/2-1; i++)
  for(j=0; j<N/2-1; j++)
  R:  B[i][j]=g(A[i][2*j], B[i][j+1],
       B[i+1][j+1], B[i][j]);
```



Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.

Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.
- A ***map*** is a binary relation from one set to another.

Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.
- A ***map*** is a binary relation from one set to another.
- The ***inverse map*** of a map M, denoted by M^{-1}, is the set of the pairs $(\vec{j}, \vec{i})$ such that $(\vec{i}, \vec{j}) \in M$.

Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.
- A ***map*** is a binary relation from one set to another.
- The ***inverse map*** of a map M, denoted by M^{-1}, is the set of the pairs $(\vec{j}, \vec{i})$ such that $(\vec{i}, \vec{j}) \in M$.
- The **domain (resp. range)** of M denoted by $\mathrm{Dom}(M)$ (resp. $\mathrm{Range}(M)$) is the set of all first elements of members of M (resp. M^{-1}).

Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.
- A ***map*** is a binary relation from one set to another.
- The ***inverse map*** of a map M, denoted by M^{-1}, is the set of the pairs $(\vec{j}, \vec{i})$ such that $(\vec{i}, \vec{j}) \in M$.
- The **domain (resp. range)** of M denoted by $\mathrm{Dom}(M)$ (resp. $\mathrm{Range}(M)$) is the set of all first elements of members of M (resp. M^{-1}).
- We denote by $\mathrm{lexmax}(M)$ **(resp.** $\mathrm{lexmin}(M)$**)** the subset of M consisting of all pairs $(\vec{i}, \vec{j})$, where $\vec{i} \in \mathrm{Dom}(M)$ and $\vec{j}$ is the lexicographically largest (resp. smallest) $\vec{k} \in \mathrm{Range}(M)$ such that $(\vec{i}, \vec{k}) \in M$.

Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.
- A ***map*** is a binary relation from one set to another.
- The ***inverse map*** of a map M, denoted by M^{-1}, is the set of the pairs $(\vec{j}, \vec{i})$ such that $(\vec{i}, \vec{j}) \in M$.
- The **domain (resp. range)** of M denoted by $\mathrm{Dom}(M)$ (resp. $\mathrm{Range}(M)$) is the set of all first elements of members of M (resp. M^{-1}).
- We denote by $\mathrm{lexmax}(M)$ **(resp.** $\mathrm{lexmin}(M)$**)** the subset of M consisting of all pairs $(\vec{i}, \vec{j})$, where $\vec{i} \in \mathrm{Dom}(M)$ and $\vec{j}$ is the lexicographically largest (resp. smallest) $\vec{k} \in \mathrm{Range}(M)$ such that $(\vec{i}, \vec{k}) \in M$.
- The ***composition*** of two maps M_1 and M_2 is denoted by $M_1(M_2)$. It is the set of all pairs $(\vec{i}, \vec{j})$, such that there exists a vector $\vec{k}$, where $(\vec{i}, \vec{k}) \in M_2$ and $(\vec{k}, \vec{j}) \in M_1$.

Background

Integer sets and maps

- ISL library represents **Z-polyhedra** as *sets* of integer tuples.
- A ***map*** is a binary relation from one set to another.
- The ***inverse map*** of a map M, denoted by M^{-1}, is the set of the pairs $(\vec{j}, \vec{i})$ such that $(\vec{i}, \vec{j}) \in M$.
- The **domain (resp. range)** of M denoted by $\mathsf{Dom}(M)$ (resp. $\mathsf{Range}(M)$) is the set of all first elements of members of M (resp. M^{-1}).
- We denote by $\mathsf{lexmax}(M)$ **(resp.** $\mathsf{lexmin}(M)$**)** the subset of M consisting of all pairs $(\vec{i}, \vec{j})$, where $\vec{i} \in \mathsf{Dom}(M)$ and $\vec{j}$ is the lexicographically largest (resp. smallest) $\vec{k} \in \mathsf{Range}(M)$ such that $(\vec{i}, \vec{k}) \in M$.
- The ***composition*** of two maps M_1 and M_2 is denoted by $M_1(M_2)$. It is the set of all pairs $(\vec{i}, \vec{j})$, such that there exists a vector $\vec{k}$, where $(\vec{i}, \vec{k}) \in M_2$ and $(\vec{k}, \vec{j}) \in M_1$.
- Given two sets S_1 and S_2, the $\mathsf{lexleset}(\mathsf{S}_1, \mathsf{S}_2)$ maps each element $\vec{i} \in S_1$ to all elements $\vec{j} \in S_2$, where $\vec{i}$ is lexicographically less or equal to $\vec{j}$.

Transformation Algorithm (1/5)

Pipeline map

Consider two statements in a program:

- $\texttt{S}$: iteration domain $\mathcal{I}$, writes in memory location $\mathcal{M}$, $\mathit{Wr}(\mathcal{I} \to \mathcal{M})$
- $\texttt{T}$: iteration domain $\mathcal{J}$, reads from memory location $\mathcal{M}$, $\mathit{Rd}(\mathcal{J} \to \mathcal{M})$

The **pipeline map** between $\texttt{S}$ and $\texttt{T}$ is $\mathcal{T}_{\texttt{S},\texttt{T}}(\mathcal{I} \to \mathcal{J})$, where $(\vec{i}, \vec{j}) \in \mathcal{T}_{\texttt{S},\texttt{T}}$ if and only if:

1. after running all iterations of $\texttt{S}$ up to $\vec{i}$, we can safely run all iterations of $\texttt{T}$ up to $\vec{j}$,
2. $\vec{i}$ is the smallest vector and $\vec{j}$ is the largest vector with Property (1).

Transformation Algorithm (2/5)

Algorithm step I, computing pipeline map and source/target blocking map

1. Relate the iteration domains:
   $$[\mathcal{P}(\mathcal{J} \to \mathcal{I}), \mathcal{P} = Wr^{-1}(Rd)], \mathsf{Domain}(\mathcal{P}) = \mathcal{D}_{\mathcal{P}}$$
2. Map each member of $\mathcal{D}_{\mathcal{P}}$ to all members that are less than or equal to it:
   $$\mathcal{D}'_{\mathcal{P}}(\mathcal{J} \to \mathcal{J})$$
3. Map each $\vec{j} \in \mathcal{J}$ to the largest $\vec{i} \in \mathcal{I}$ that $\vec{j}$ and its previous iterations depend on:
   $$[\mathcal{H}(\mathcal{J} \to \mathcal{I}), \mathcal{H} = \mathsf{lexmax}(\mathcal{P}(\mathcal{D}'))]$$
4. The pipeline map is:
   $$\mathcal{T}_{\mathtt{S},\mathtt{T}} = \mathsf{lexmax}(\mathcal{H}^{-1})$$
5. Partition iteration domain of S (T) with the domain (range) of $\mathcal{T}_{\mathtt{S},\mathtt{T}}$:
   $$\mathcal{B} = \mathsf{Dom}(\mathcal{T}_{\mathtt{S},\mathtt{T}}), \mathcal{B}' = \mathsf{lexleset}(\mathcal{I}, \mathcal{B}), (\mathcal{B} = \mathsf{Range}(\mathcal{T}_{\mathtt{S},\mathtt{T}})\ \mathcal{B}' = \mathsf{lexleset}(\mathcal{J}, \mathcal{B}))$$
6. Compute **source (target) blocking map**:
   $$[\mathcal{V}_{\mathtt{S}}(\mathcal{I} \to \mathcal{I}), \mathsf{lexmin}(\mathcal{B}')], ([\mathcal{Y}_{\mathtt{T}}(\mathcal{J} \to \mathcal{J}), \mathsf{lexmin}(\mathcal{B}')])$$

Transformation Algorithm (3/5)

After finding the pipeline maps between all pairs of dependent statements, we use them to block the iteration domains and construct **pipeline blocking maps**.

The final blocks are such that:

- each block is an atomic task,
- we can establish a pipeline relation between all blocks of all statements,
- maximize the number of blocks of different loops that can execute in parallel.

In the last step, we find **dependency relations** between the tasks.

Transformation Algorithm (4/5)

Algorithm step II, computing pipeline blocking maps

There are several source and target blocking maps associated with each statement.

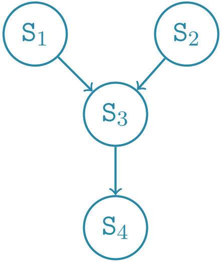
- Minimize the size of the blocks and construct the **optimal blocks**.
- get the lexmin of the union of all source and target blocking maps:
  $\mathcal{E}_{\mathsf{S}} = \mathsf{lexmin}((\bigcup_j(\mathcal{V}_{\mathsf{S}}^j) \cup (\bigcup_i(\mathcal{Y}_{\mathsf{S}}^i)))$

Algorithm step III, computing pipeline dependency relations

In a task-parallel program, there are dependency relations between different tasks.

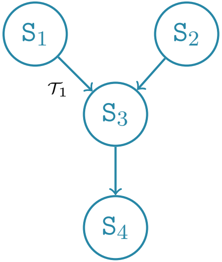
- **Pipeline dependency relations** map each block to the blocks it needs to run correctly.
- For a statement S and a pipeline map $\mathcal{T}_i$, where S is the target:
  $\mathcal{Q}_{\mathsf{S}}^i = \mathcal{T}_i^{-1}(\mathcal{Y}_i(\mathsf{Range}(\mathcal{E}_{\mathsf{S}})))$

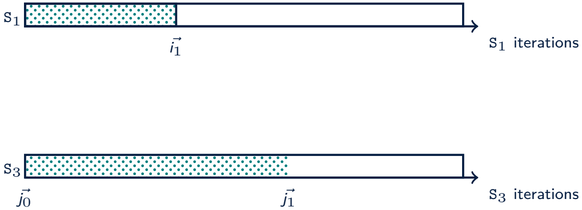
Transformation Algorithm (5/5)



skip slide

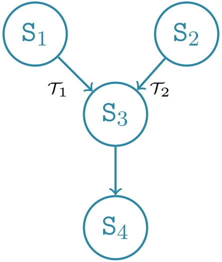
Transformation Algorithm (5/5)

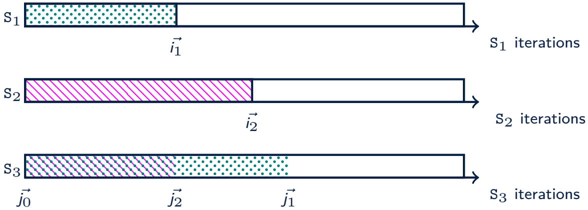




skip slide

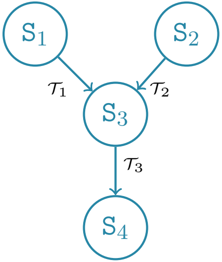
Transformation Algorithm (5/5)

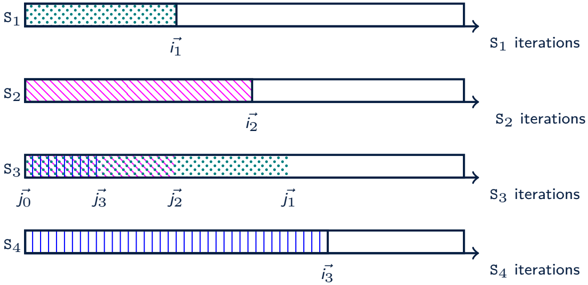




skip slide

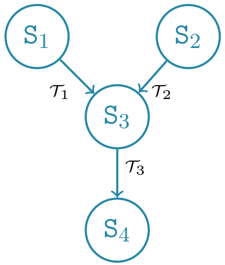
Transformation Algorithm (5/5)

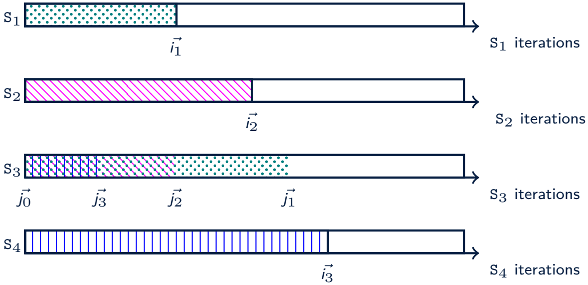




skip slide

Transformation Algorithm (5/5)





Optimal block of S_3: $\langle S_3, \vec{j_3} \rangle$
Pipeline dependencies: $\langle S_1, \vec{i_1} \rangle$, $\langle S_2, \vec{i_2} \rangle$

skip slide

Implementation (1/2)

Analysis passes of Polly

Extend analysis passes of Polly to compute pipeline information for the iteration domains.

Scheduling

1. Create a schedule tree to iterate **over** blocks,
2. Create a schedule tree to iterate **inside** each blocks,
3. **Expand** the first tree with the second tree.
4. Create `pw_multi_aff_list` objects from pipeline dependency relations,
5. Add the `pw_multi_aff_list` objects as mark nodes to the schedule tree.

Implementation (2/2)

Abstract syntax tree

Generate AST from the new schedule tree.
The mark nodes in the schedule tree **annotates** the AST.

Code generation

1. Outline tasks to function calls,
2. Compute unique integer numbers from `pw_multi_aff_list` objects
   - this can be used in OpenMP `depend` clauses.
3. Replace the tasks part in the code with call to the `CreateTask` function that:
   - gets tasks and dependencies, creates OpenMP `tasks` with proper `depend` clauses,
   - handles the order between tasks created from the same loop nest.

Evaluation

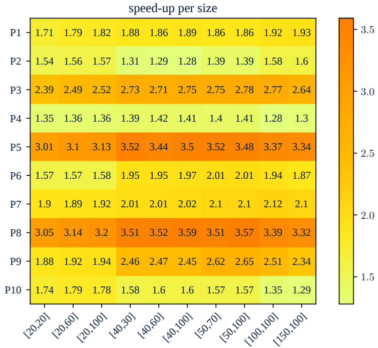


Figure: Speed-up of the tests with different access functions, considering different sizes, comparing sequential version and pipelined version.

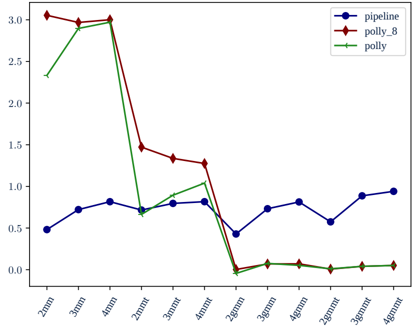


Figure: Comparing logarithm of speed-up gains of Polly running by all available threads, Polly running by `n` threads (`n` is the number of loop nests), and cross-loop pipelining for variants of generalized matrix multiplication.

References I

Rui-Juan Jing, Marc Moreno Maza, Yan-Feng Xie, and Chun-Ming Yuan.
Efficient detection of redundancies in systems of linear inequalities.
In Jonathan D. Hauenstein, Wen-shin Lee, and Shaoshi Chen, editors, *Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation, ISSAC 2024, Raleigh, NC, USA, July 16-19, 2024*, pages 351–360. ACM, 2024.

Marc Moreno Maza and Linxiao Wang.
Computing the integer hull of convex polyhedral sets.
In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *CASC 2022, Proceedings*, volume 13366 of *Lecture Notes in Computer Science*, pages 246–267. Springer, 2022.

References II

Delaram Talaashrafi, Johannes Doerfert, and Marc Moreno Maza.
A pipeline pattern detection technique in polly.
In *Workshop Proceedings of the 51st International Conference on Parallel Processing, ICPP Workshops 2022, Bordeaux, France, 29 August 2022 - 1 September 2022*, pages 18:1–18:10. ACM, 2022.

L. Khachiyan.
Fourier-motzkin elimination method.
In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization, Second Edition*, pages 1074–1077. Springer, 2009.

H. Greenberg.
Consistency, redundancy, and implied equalities in linear systems.
Annals of Mathematics and Artificial Intelligence, 17:37–83, 1996.

References III

E. Balas.
Projection with a minimal system of inequalities.
Computational Optimization and Applications, 10(2):189–193, 1998.

T. Huynh, C. Lassez, and J. L. Lassez.
Practical issues on the projection of polyhedral sets.
Annals of mathematics and artificial intelligence, 6(4):295–315, 1992.

J. L. Lassez.
Parametric queries, linear constraints and variable elimination.
In *International Symposium on Design and Implementation of Symbolic Computation Systems*, pages 164–173. Springer, 1990.

R. J. Jing, M. Moreno-Maza, and D. Talaashrafi.
Complexity estimates for fourier-motzkin elimination.
In *Proceedings of CASC*, pages 282–306. Springer, 2020.

Thank You!