

# Splitting Polyhedra to Generate More Efficient Code

Efficient Code Generation in the Polyhedral Model is Harder Than We Thought

January 23rd 2017, IMPACT'17, HiPEAC, Stockholm, Sweden

---

Harenome Razanajato, Vincent Loechner, Cédric Bastoul

University of Strasbourg



# Introduction to the $\mathbb{Z}$ -Polyhedral Model

The polyhedral model – also known as the polytope model – is a mathematical abstraction that can be used to represent, manipulate and optimize loop nests.

## Definition

A parametric polyhedron  $P_p$  with  $n$  parameters is a set of  $d$ -dimensional vectors  $x$ :

$$P_p = \{x \in \mathbb{K}^d \mid Ax \geq Bp + c\}$$

where  $A \in \mathbb{K}^{m \times d}$ ,  $B \in \mathbb{K}^{m \times n}$  and  $c \in \mathbb{K}^m$  so that for each fixed value  $p_0$ ,  $P_{p_0}$  defines a polyhedron in  $\mathbb{K}^d$ .

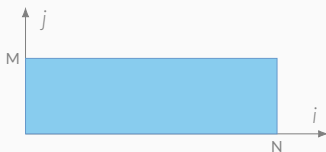
# Introduction to the $\mathbb{Z}$ -Polyhedral Model

```
1  for (i = 0; i <= N; ++i)
2    for (j = 0; j <= M; ++j)
3      S(i, j);
```

(a) Loop nest

$$\mathcal{S}(i,j) = \left\{ (i,j) \in \mathbb{Z}^2 \left| \begin{array}{l} 0 \leq i \leq N \\ 0 \leq j \leq M \end{array} \right. \right\}$$

(b) Constraints



(c) 2D representation

**Figure 1:** Various representations of a loop nest in the polyhedral model

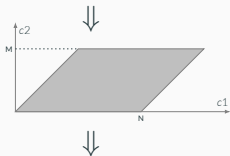
# Overview of the Polyhedral Model Framework

```
1  for (i = 0; i <= N; ++i)
2    for (j = 0; j <= M; ++j)
3      S(i, j);
```

1 Raising



2 Transformation



3 Code generation

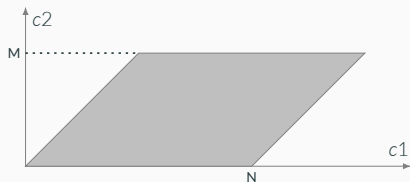
```
1  for (c1 = 0; c1 <= N+M; c1++)
2    for (c2 = max(0, c1-N); c2 <= min(M, c1); c2++)
3      S1(c1, c2);
```

# A Simple Polyhedron

Let's consider the following polyhedron:

$$\mathcal{P}(N, M) = \left\{ (c1, c2) \in \mathbb{Z}^2 \left| \begin{array}{l} 0 \leq c1 \leq N + M \\ 0 \leq c2 \\ c1 - N \leq c2 \\ c2 \leq c1 \\ c2 \leq M \end{array} \right. \right\}$$

(a) Constraints

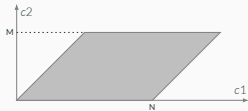


(b) 2D representation

**Figure 2:** A simple polyhedron

What code would you generate?

# Code Generation is Easy!



Most tools would generate this:

```
1 for (c1 = 0; c1 <= N+M; c1++)  
2     for (c2 = max(0, c1-N); c2 <= min(M, c1); c2++)  
3         S1(c1, c2);
```

**Listing 1:** Code for scanning the polyhedron  $\mathcal{P}$  from Figure 2

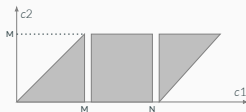
...and most people would be happy.

Are you?

# Is Efficient Code Generation Easy?

What about this:

```
1  for (c1 = 0; c1 <= M; c1++)
2      for (c2 = 0; c2 <= c1; c2++)
3          S1(c1, c2);
4  for (c1 = M+1; c1 <= N-1; c1++)
5      for (c2 = 0; c2 <= M; c2++)
6          S1(c1, c2);
7  for (c1 = N; c1 <= N+M; c1++)
8      for (c2 = c1-N; c2 <= M; c2++)
9          S1(c1, c2);
```



**Listing 2:** Another way to scan the polyhedron  $\mathcal{P}$  from Figure 2



# Quick Comparison

```
1  for (c1 = 0; c1 <= N+M; c1++)
2      for (c2 = max(0, c1-N); c2<= min(M, c1); c2++)
3          S1(c1, c2);
```

```
1  for (c1 = 0; c1 <= M; c1++)
2      for (c2 = 0; c2 <= c1; c2++)
3          S1(c1, c2);
4  for (c1 = M+1; c1 <= N-1; c1++)
5      for (c2 = 0; c2 <= M; c2++)
6          S1(c1, c2);
7  for (c1 = N; c1<= N+M; c1++)
8      for (c2 = c1-N; c2<= M; c2++)
9          S1(c1, c2);
```

- high control overhead
- small code size
- fast code generation

- low control overhead
- increased code size
- slow code generation
- speedups reach 1.32 with gcc, 1.67 with icc and 4.63 with clang

Our goals:

- eliminate min and max computations in order to reduce control overhead
- help the compiler:
  - polyhedra splitting can ease vectorization and branch prediction
  - tiled iteration domains can benefit from polyhedra splitting
  - different splitting strategies impact the overall quality of generated code, code size and generation time

# Outline

Introduction

Splitting polyhedra

Prerequisites

Splitting Polyhedra in QRW

Experimental results

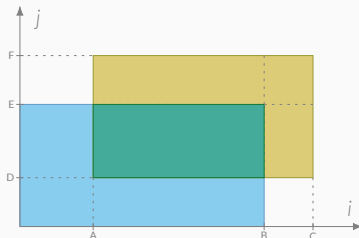
Conclusion

# Reminder on CLooG

- CLooG: Chunky Loop Generator
- Generates code for scanning  $\mathbb{Z}$ -polyhedra
- Extended version of the Quilleré, Rajopadhye, and Wilde algorithm [2]

# CLooG's extended QRW Algorithm

1. Intersect the polyhedra with the context
2. Project the polyhedra onto the outermost dimension
3. Separate the projections into a disjunct list of polyhedra
4. (Compute lexicographic order and strides)
5. Recurse on the inner dimensions



$$\text{Context} : \left\{ \begin{array}{l} 0 \leq A < B < C \\ 0 \leq D < E < F \end{array} \right\}$$

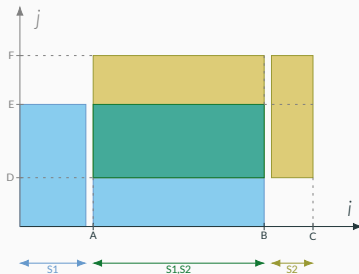
$$\mathcal{S}_1(B, E) = \left\{ (i, j) \in \mathbb{Z}^2 \mid \begin{array}{l} 0 \leq i \leq B \\ 0 \leq j \leq E \end{array} \right\}$$

$$\mathcal{S}_2(A, C, D, F) = \left\{ (i, j) \in \mathbb{Z}^2 \mid \begin{array}{l} A \leq i \leq C \\ D \leq j \leq F \end{array} \right\}$$

Figure 5: Intersection of the polyhedra with the context

# CLooG's extended QRW Algorithm

1. Intersect the polyhedra with the context
2. Project the polyhedra onto the outermost dimension
3. Separate the projections into a disjunct list of polyhedra
4. (Compute lexicographic order and strides)
5. Recurse on the inner dimensions

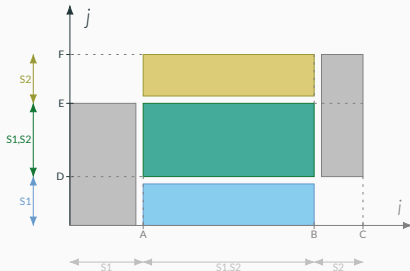


```
1  for (i = 0; i <= A-1; ++i)
2    /* S1 */
3  for (i = A; i <= B; ++i) {
4    /* S1 and S2 */
5  }
6  for (i = B+1; i <= C; ++i)
7    /* S2 */
```

Figure 6: Projection and separation on the first dimension

# CLooG's extended QRW Algorithm

1. Intersect the polyhedra with the context
2. Project the polyhedra onto the outermost dimension
3. Separate the projections into a disjunct list of polyhedra
4. (Compute lexicographic order and strides)
5. Recurse on the inner dimensions



```
1  for (i = 0; i <= A-1; ++i)
2    for (j = 0; j <= E; ++j)
3      S1(i, j);
4  for (i = A; i <= B; ++i) {
5    for (j = 0; j <= D-1; ++j)
6      S1(i, j);
7    for (j = D; j <= E; ++j) {
8      S1(i, j);
9      S2(i, j);
10   }
11   for (j = E+1; j <= F; ++j)
12     S2(i, j);
13 }
14 for (i = B+1; i <= C; ++i)
15   for (j = D; j <= F; ++j)
16     S2(i, j);
```

Figure 7: Recursion on the second dimension

# Chamber Decomposition

- Chamber : a part of the parameter domain in which all vertices coordinates are affine functions of parameters
- Assuming the number of parameters of a parameterized polyhedron is  $m$ , the validity domains are obtained by computing the  $m$ -faces of the polyhedron using Loechner and Wilde's algorithm [1]
- The PolyLib allows to compute the validity domains of parameterized polyhedra



# Splitting Polyhedra in QRW

Introduction

Splitting polyhedra

Prerequisites

Splitting Polyhedra in QRW

Experimental results

Conclusion

# Splitting polyhedra in CLooG's extended QRW

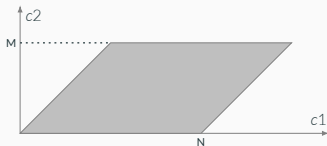
- After the separation step of CLooG's algorithm
- Relies on chamber decomposition
- Previous parameters, outer loop indices and the current loop index must be considered as parameters

# Example i

1. Let's consider the polyhedron from Figure 2 again:

$$\mathcal{P}(N, M) = \left\{ (c_1, c_2) \in \mathbb{Z}^2 \left| \begin{array}{l} 0 \leq c_1 \leq N + M \\ 0 \leq c_2 \\ c_1 - N \leq c_2 \\ c_2 \leq c_1 \\ c_2 \leq M \end{array} \right. \right\}$$

(a) Constraints



(b) 2D representation

$$\mathcal{P}_{c_1}(N, M, c_1) = \left\{ (c_2) \in \mathbb{Z} \left| \begin{array}{l} 0 \leq c_2 \\ c_1 - N \leq c_2 \\ c_2 \leq c_1 \\ c_2 \leq M \end{array} \right. \right\}$$

(c) Considering  $c_1$  as a parameter

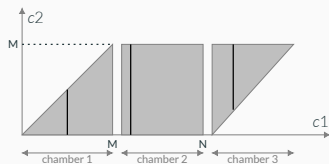
# Example ii

2. Computing the chambers of the parametric polyhedron  $\mathcal{P}_{c_1}$ :

$$\mathcal{C}_1 = \{(N, M, c_1) \in \mathbb{Z}^3 \mid 0 \leq c_1 \leq M\}$$

$$\mathcal{C}_2 = \{(N, M, c_1) \in \mathbb{Z}^3 \mid M + 1 \leq c_1 \leq N - 1\}$$

$$\mathcal{C}_3 = \{(N, M, c_1) \in \mathbb{Z}^3 \mid N \leq c_1 \leq N + M\}$$

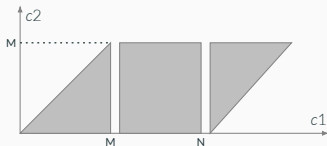


3. Revert  $c_1$  to an iterator:

$$\mathcal{P}_1(N, M) = \left\{ (c_1, c_2) \in \mathbb{Z}^2 \mid \begin{array}{l} 0 \leq c_1 \leq M \\ 0 \leq c_2 \leq c_1 \end{array} \right\}$$

$$\mathcal{P}_2(N, M) = \left\{ (c_1, c_2) \in \mathbb{Z}^2 \mid \begin{array}{l} M+1 \leq c_1 \leq N-1 \\ 0 \leq c_2 \leq M \end{array} \right\}$$

$$\mathcal{P}_3(N, M) = \left\{ (c_1, c_2) \in \mathbb{Z}^2 \mid \begin{array}{l} N \leq c_1 \leq N+M \\ c_1 - N \leq c_2 \leq M \end{array} \right\}$$



# Experimental Results

---

Introduction

Splitting polyhedra

Experimental results

Conclusion

Comparing the results of the mainline CLoog and our PolyLib-enhanced version:

- CLoog's test suite and PolyBench 4.1
- Code scheduled with PLUTO 0.11.4 (option `--tile`)
- Code compiled using `gcc 6.2.1`, `clang 3.8.1` and `icc 17.0.0` with options `-O0` and `-O3 -march=native`
- Tested on a 2.40GHz Intel Xeon E5-2620v3, running linux 4.8.

# Polybenches -O0

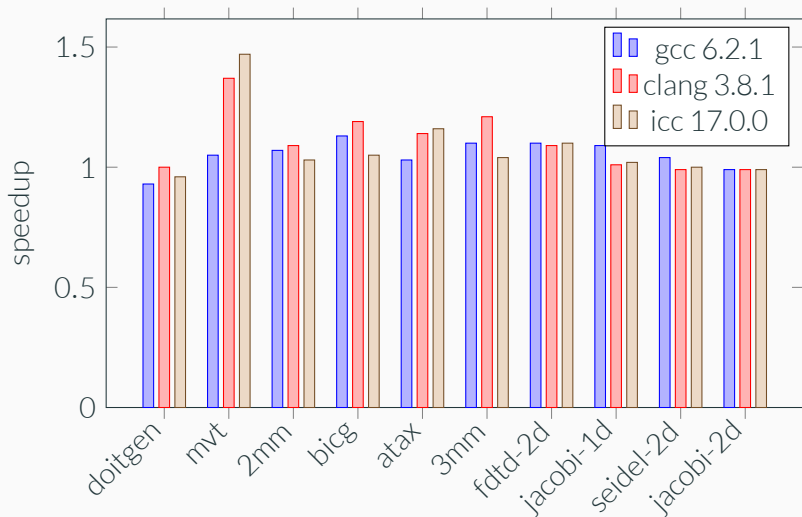


Figure 11: Speedups on the PolyBench test suite with -O0



# Polybenches -O3 -march=native

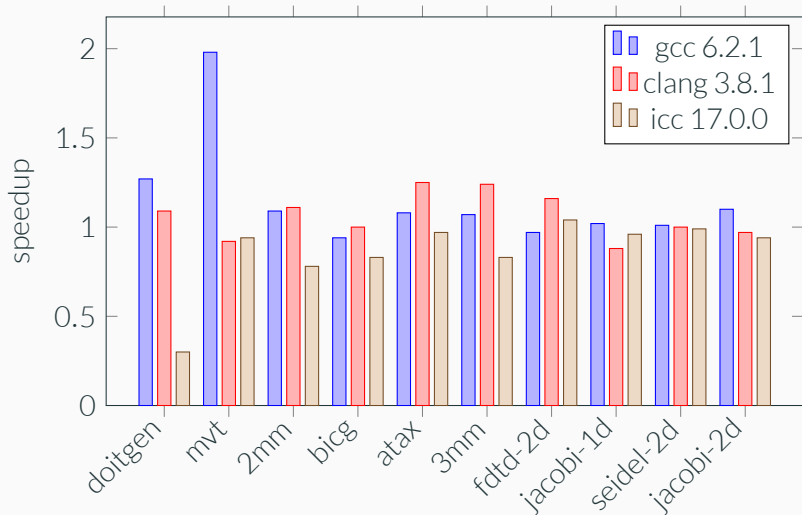


Figure 12: Speedups on the PolyBench tests with -O3 -march=native

# Speedups on CLooG's Test Suite

compiler	speedup > 1			speedup < 1			geomean speedup
	percentage	maximum	geomean	percentage	minimum	geomean	
gcc-O0	59	1.34	1.07	41	0.81	0.96	1.02
icc-O0	66.7	1.21	1.03	33.3	0.97	0.99	1.01
clang-O0	82.5	1.96	1.19	17.5	0.5	0.9	1.12
gcc-O3	51.2	1.07	1.03	48.8	0.9	0.97	1.00
icc-O3	61.5	1.11	1.02	38.5	0.89	0.98	1.00
clang-O3	63.9	1.17	1.09	36.1	0.66	0.95	1.03

**Figure 13:** Overview of the speedups for the CLooG tests

- Slower CLoog code generation
  - 5 times slower for CLoog's test suite
  - 10 times slower for the PolyBench
- Increased code size (geometric mean growth of 257%)

# Conclusion

Introduction

Splitting polyhedra

Experimental results

Conclusion

# Conclusion

- A new method for splitting polyhedra in order to reduce control overhead
- Splitting polyhedra *may* improve the efficiency of the generated code
- Slower code generation and increased code size
- Future work: determining when such splitting should or should not be performed
- Efficient code generation is harder than we thought!

Thank you!



Vincent Loechner and Doran K. Wilde.

**Parameterized Polyhedra and Their Vertices.**

*International Journal of Parallel Programming*, 25(6):525–549,  
December 1997.



Fabien Quilleré, Sanjay Rajopadhye, and Doran Wilde.

**Generation of efficient nested loops from polyhedra.**

*International Journal of Parallel Programming*, 28(5):469–498,  
2000.