Compilation and Optimization of Static Control Flow Programs: Polyhedral Compilation at Large

Louis-Noël Pouchet
Associate Professor, Computer Science Department
Joint appointment in Electrical and Computer Engineering
pouchet@colostate.edu

13th International Workshop on Polyhedral Compilation Techniques

IMPACT'23

January 16th, 2023

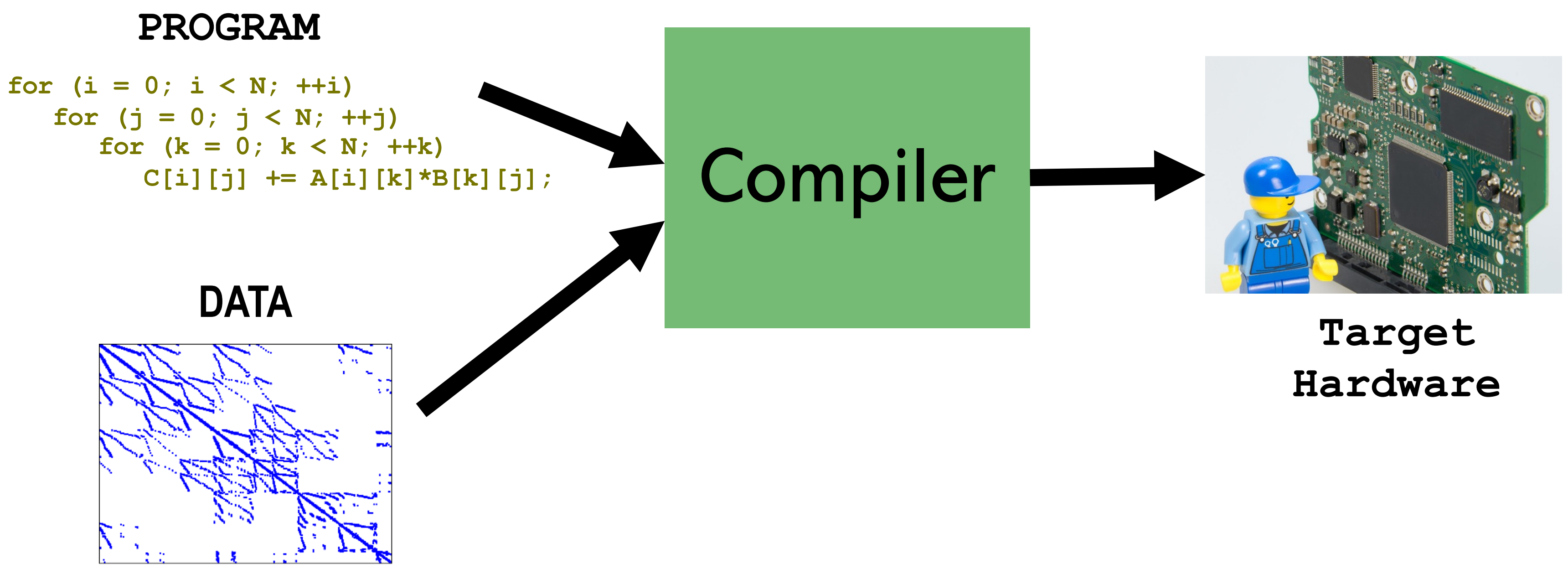
A Path to Polyhedral Compilation…

- Born in 1980 in France
- …
- 2001-03: 2 years preparatory classes at EPITA
- 2003: Start 3-years engineering school (CS focused)
- 2004: Start undergrad research in automata theory
- 2005: Decided I wanted to do a (French) PhD, designing medical devices to assist disabled people, so… med school or…
- 2005: Enroll concurrently in a MS at University Paris XI, on artificial learning and neurobiology
- 2006: MS/EPITA final internship time (6 months), went to INRIA to work on compilers!
- 2006-2009: PhD at INRIA
- 2010-2012: Postdoc at Ohio State University with P. Sadayappan, on HPC techniques
- 2012-2014: Visiting Assistant Prof. at UCLA, focus on FPGAs and high-level synthesis
- 2014-2016: Research Assistant Prof. at OSU
- Since 2016: at CSU

Disclaimer about the Content of this Talk

- **It will be (very) partial, biased and oriented much around work we did with colleagues**
- **This is NOT a complete survey of the successes and limitations of polyhedral compilation! It would be too long ☺**
- **I will NOT talk about so many impactful work in our community**
  - Foundational algorithms and results
  - Tools and their implementation
- **The work presented here would not have been possible without the foundations and tools (un)referenced above! ☺**

Outline of this Talk

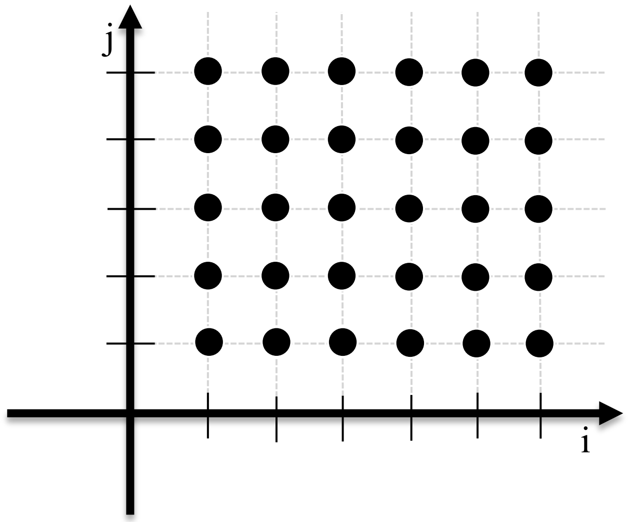


```
for (i = 0; i < N; ++i)
   for (j = 0; j < N; ++j)
      for (k = 0; k < N; ++k)
         C[i][j] += A[i][k]*B[k][j];
```

- **Objective for today: pave a way of some uses of Polyhedral Compilation!**
- **Outline: first some background, then…**
  - A word on representing **programs**, using domain-specific languages
  - And **data**, using union of polyhedral,
  - Some **optimization** tools
  - Representing **hardware designs** as polyhedral programs, and proving their equivalence

But What is a Polyhedron?

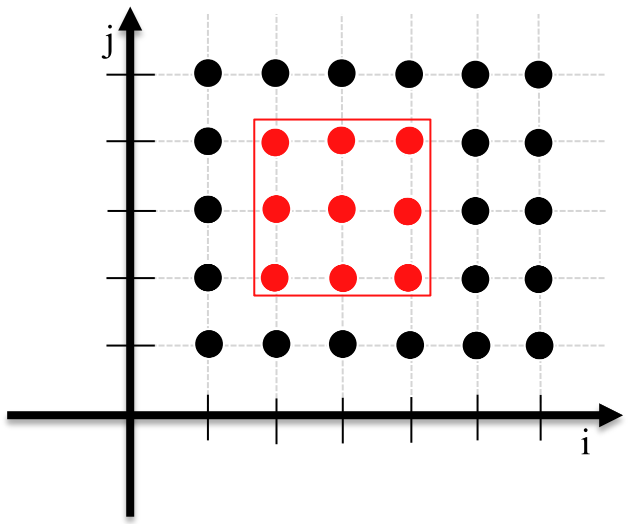
Example



Grid of 2D Integer points

But What is a Polyhedron?

Example



2D Integer points

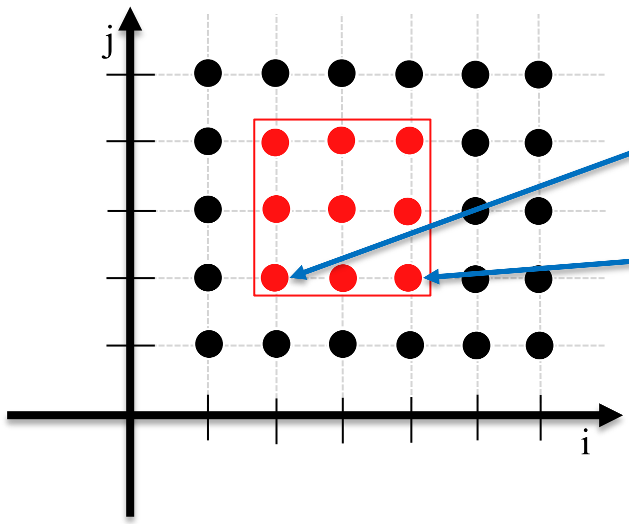
List of points

i	j
2	2
2	3
2	4
3	2
3	3
3	4
4	2
4	3
4	4

Compact description

But What is a Polyhedron?

Example



2D Integer points

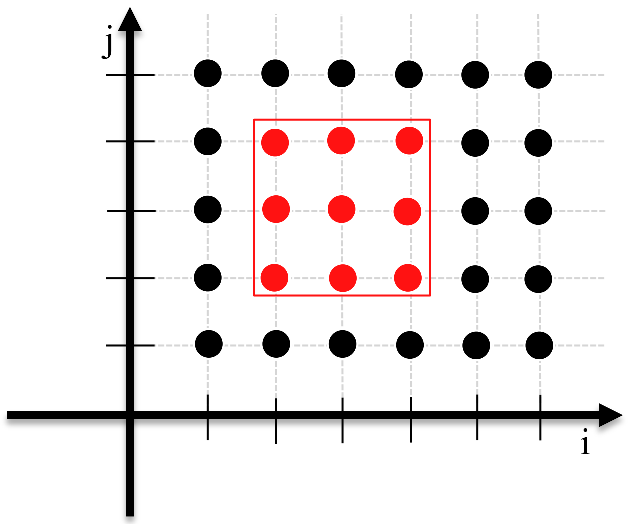
List of points

i	j
2	2
2	3
2	4
3	2
3	3
3	4
4	2
4	3
4	4

Compact description

But What is a Polyhedron?

Example



2D Integer points

List of points

i	j
2	2
2	3
2	4
3	2
3	3
3	4
4	2
4	3
4	4

Compact description

$$D : \{ [i,j] : 2 \leq i \leq 4 \text{ and } 2 \leq j \leq 4 \}$$

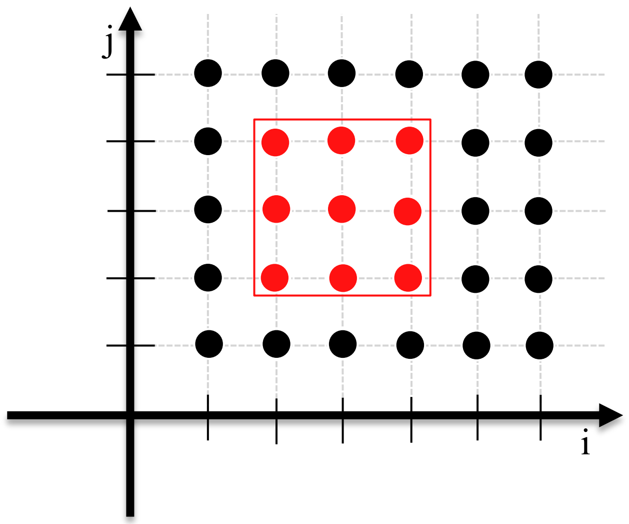
Polyhedron: described as the intersection of half-planes (e.g., $i \leq 2$), all points in the intersection are in the polyhedron

Dimensionality: 2

In this work: model only polyhedra of integer points

But What is a Polyhedron?

Example



2D Integer points

List of points

i	j
2	2
2	3
2	4
3	2
3	3
3	4
4	2
4	3
4	4

Compact description

$$D : \{ [i,j] : 2 \leq i \leq 4 \text{ and } 2 \leq j \leq 4 \}$$

Polyhedron: described as the intersection of half-planes (e.g., $i \leq 2$), all points in the intersection are in the polyhedron

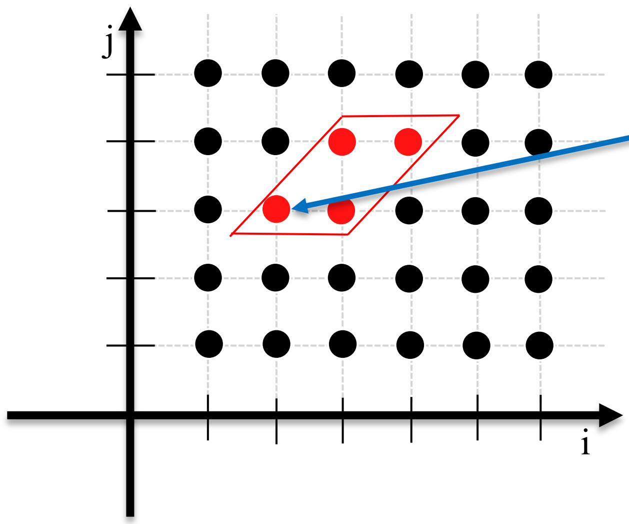
Dimensionality: 2

In this work: model only polyhedra of integer points

More complex shapes?

But What is a Polyhedron?

Example



2D Integer points

List of points

i	j
2	3
3	3
3	4
4	4

Compact description

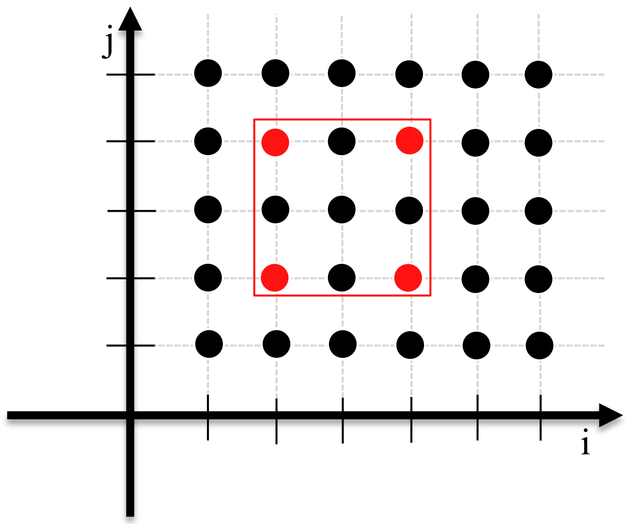
$$D : \{ [i,j] : 2 \leq i \leq 4 \text{ and } 3 \leq j \leq 4 \text{ and } j \geq i \text{ and } j \leq i+1 \}$$

Polyhedron: possibly many half planes to describe it => **affine inequalities**

Inequalities may involve several variables / dimensions

But What is a Polyhedron?

Example



2D Integer points

List of points

i	j
2	2
2	4
4	2
4	4

Compact description

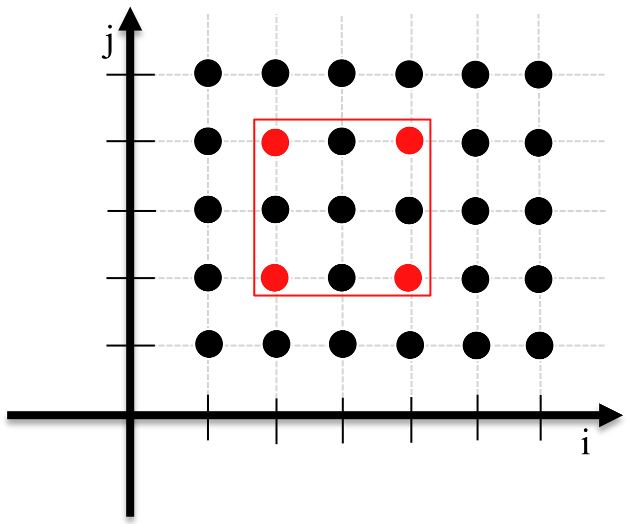
~~D : { [i,j] : $2 \le i \le 4$ and $2 \le j \le 4$ }~~

Still describes 9 points!!

But what about holes in the shape?

But What is a Polyhedron?

Example



2D Integer points

List of points

i	j
2	2
2	4
4	2
4	4

Compact description

D : { [i,j] : $1 \leq i \leq 2$ and
$1 \leq j \leq 2$ }

+

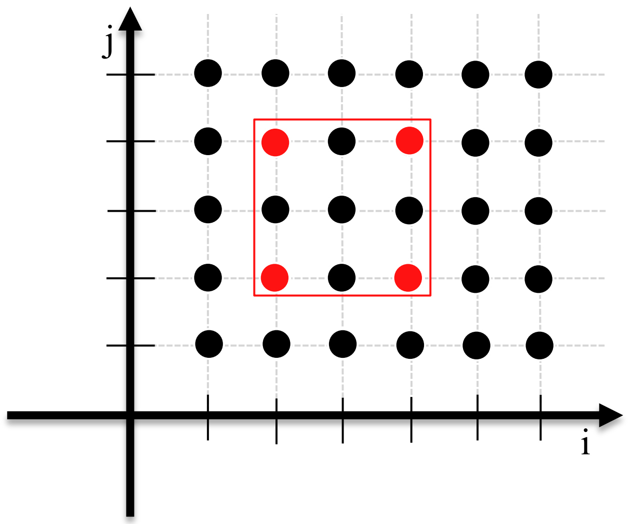
Intersected with an integer lattice:
L : { [i,j] —> [x,y] : x = 2i and y = 2j }

D contains 4 points, the lattice L captures their exact coordinates (stride of 2 here)

A polyhedron intersected with a lattice is a Z-Polyhedron

But What is a Polyhedron?

Example



2D Integer points

List of points

i	j
2	2
2	4
4	2
4	4

Compact description

D : { [i,j] : $1 \leq i \leq 2$ and $1 \leq j \leq 2$ }

+

Intersected with an integer lattice:

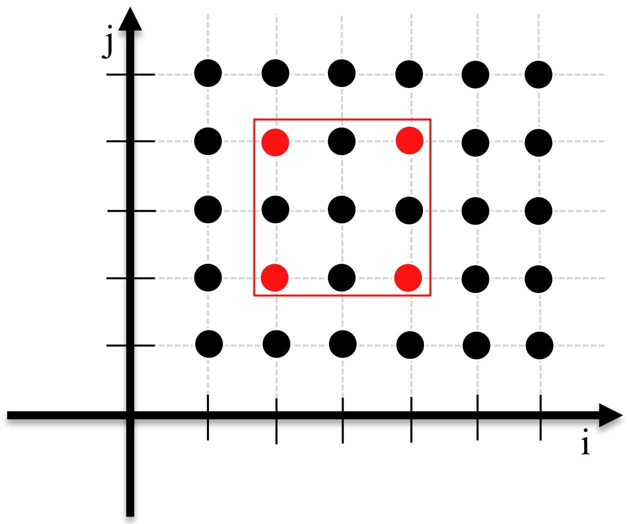
L : { [i,j] —> [x,y] : x = 2i and y = 2j }

D contains 4 points, the lattice L captures their exact coordinates (stride of 2 here)

Z-Polyhedra can have "holes", needed for "sparse" structures

Z-Polyhedra are Code, Too

Example



2D Integer points

List of points

i	j
2	2
2	4
4	2
4	4

Compact description

D : { [i,j] : $1 \leq i \leq 2$ and
$1 \leq j \leq 2$ }

+

Intersected with an integer lattice:

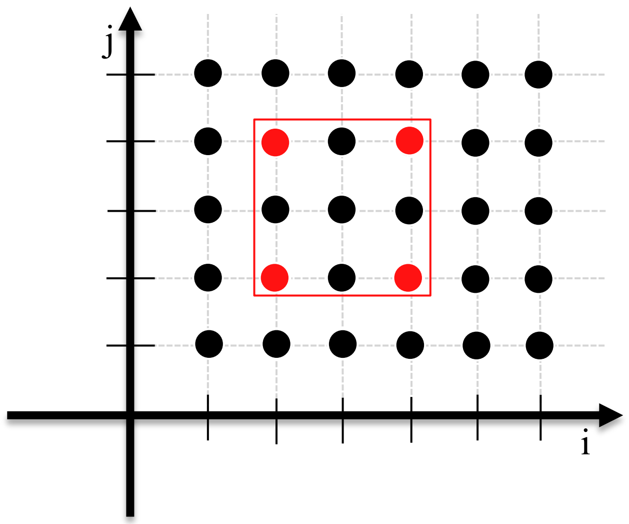
L : { [i,j] —> [x,y] : x = 2i and y = 2j }

```
for (i = 1; i <= 2; i++)
  for (j = 1; j <= 2; j++)
    S(2i,2j); // x = 2i, y = 2j
```

This code traverses all and only points in the Z-polyhedron

Z-Polyhedra are Code, Too

Example



2D Integer points

List of points

i	j
2	2
2	4
4	2
4	4

Compact description

D : { [i,j] : $1 \leq i \leq 2$ and $1 \leq j \leq 2$ }

+

Intersected with an integer lattice:

L : { [i,j] —> [x,y] : x = 2i and y = 2j }

```
for (i = 1; i <= 2; i++)
  for (j = 1; j <= 2; j++)
    S(2i,2j); // x = 2i, y = 2j
```

This code traverses all and only points in the Z-polyhedron

The 3 Stages of Polyhedral Optimization

1. **Analysis: from code to model**
   - Existing tools
     - PET, PolyOpt
     - URUK, Omega, ChiLL, PoCC
   - GCC GRAPHITE, LLVM Polly (now in mainstream)
   - Reservoir Labs R-Stream, IBM XL/Poly
2. **Transformation in the model**
   - Build and select a program transformation
3. **Code generation: from model to code**
   - "Apply" the transformation in the model
   - Regenerate syntactic (AST-based) code

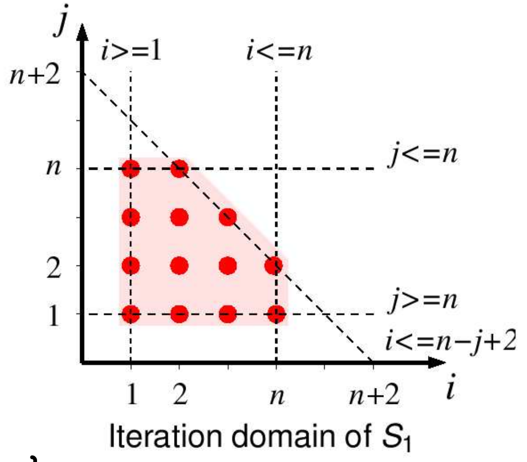
The Polyhedral Model in a Nutshell

Affine program regions:

- **Loops have affine control only (over-approximation otherwise)**
- **Iteration domain: represented as integer polyhedra**

```
for (i=1; i<=n; ++i)
. for (j=1; j<=n; ++j)
. . if (i<=n-j+2)
. . . s[i] = ...
```

$$\mathcal{D}_{S1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 2 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \geq \vec{0}$$



Iteration domain of S_1

```
D := [n] -> { [i,j] : 1 <= i <= n and
                      1 <= j <= n and i <= n-j+2 }
```

The Polyhedral Model in a Nutshell

Affine program regions:

- **Loops have affine control only (over-approximation otherwise)**
- **Iteration domain: represented as integer polyhedra**
- **Array index functions: represented as functions of the loop iterators and parameters**

$$f_{\mathbf{s}}(\vec{x_{S2}}) = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \end{array}\right] . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

```
for (i=0; i<n; ++i) {
. s[i] = 0;
. for (j=0; j<n; ++j)
. . s[i] = s[i]+a[i][j]*x[j];
}
```

$$f_{\mathbf{a}}(\vec{x_{S2}}) = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right] . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{x}}(\vec{x_{S2}}) = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \end{array}\right] . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

```
Fs := [n] -> { [i,j] -> s[i] }
Fa := [n] -> { [i,j] -> a[i][j] }
```

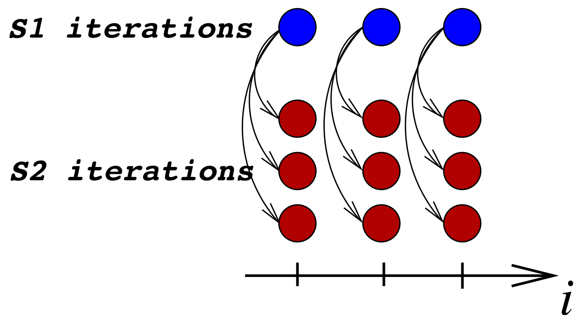
The Polyhedral Model in a Nutshell

Affine program regions:

- Loops have affine control only (over-approximation otherwise)
- Iteration domain: represented as integer polyhedra
- Array index functions: represented as functions of the loop iterators and parameters
- Data dependence between S1 and S2: a subset of the Cartesian product of *DS*1 and *DS*2 (exact analysis possible)

```
for (i=1; i<=3; ++i) {
.  s[i] = 0;
.  for (j=1; j<=3; ++j)
.  .  s[i] = s[i] + 1;
}
```

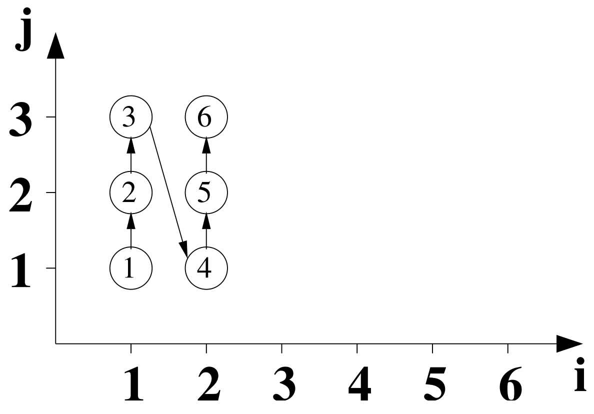
$$\mathcal{D}_{S1\delta S2} : \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ \hline 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 3 \end{array}\right] . \begin{pmatrix} i_{S1} \\ i_{S2} \\ j_{S2} \\ 1 \end{pmatrix} \begin{array}{l} = 0 \\ \geq \vec{0} \end{array}$$



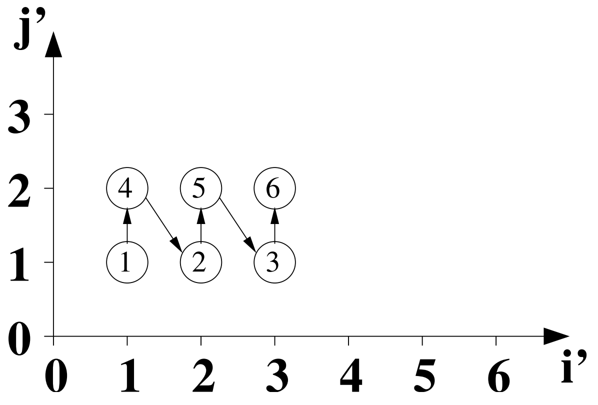
Affine Transformations

Interchange Transformation

The transformation matrix is the identity with a permutation of two rows.



$\Longrightarrow$



$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron

$A\vec{x}+\vec{a} \geq \vec{0}$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function

$\vec{y} = T\vec{x}$

$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron

$(AT^{-1})\vec{y}+\vec{a} \geq \vec{0}$

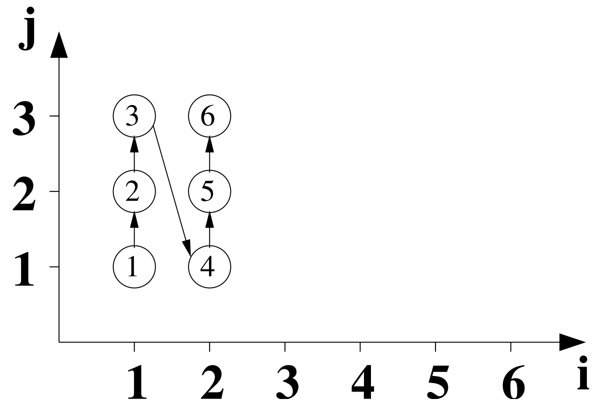
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = 1, 3
  do j' = 1, 2
     S(i=j',j=i')
```

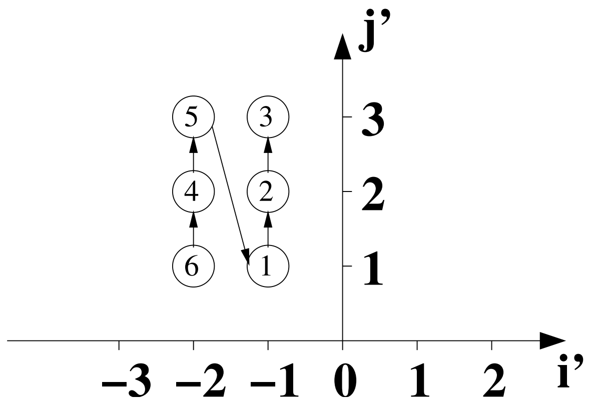
Affine Transformations

Reversal Transformation

The transformation matrix is the identity with one diagonal element replaced by -1.



$\Longrightarrow$



$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron

$A\vec{x} + \vec{a} \geq \vec{0}$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function

$\vec{y} = T\vec{x}$

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron

$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

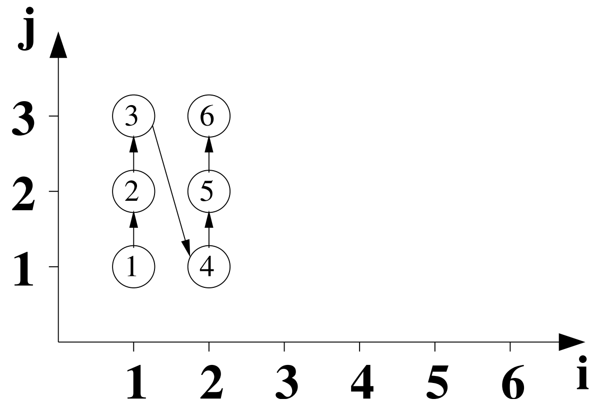
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = -1, -2, -1
  do j' = 1, 3
     S(i=3-i',j=j')
```

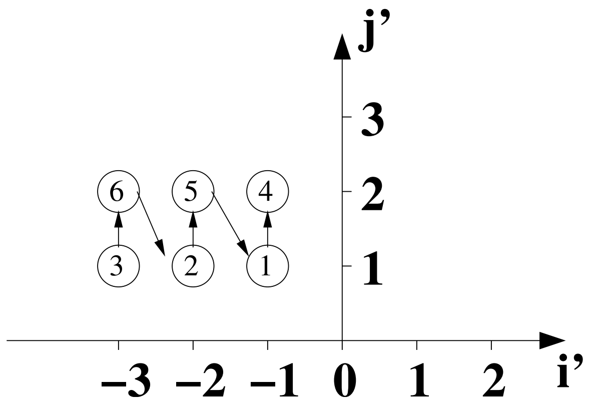
Affine Transformations

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$\Longrightarrow$



$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$

$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

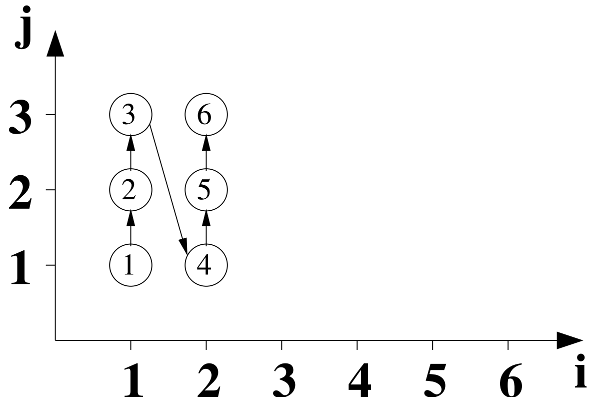
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do j' = -1, -3, -1
  do i' = 1, 2
     S(i=4-j',j=i')
```

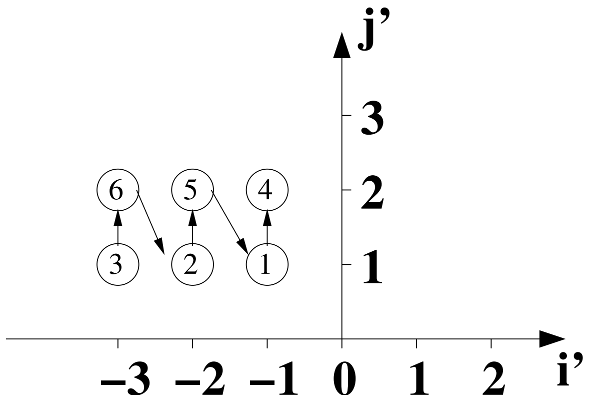
Affine Transformations

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$\Longrightarrow$



$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron

$A\vec{x}+\vec{a} \geq \vec{0}$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function

$\vec{y} = T\vec{x}$

$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron

$(AT^{-1})\vec{y}+\vec{a} \geq \vec{0}$

```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do j' = -1, -3, -1
  do i' = 1, 2
     S(i=4-j',j=i')
```

The Polyhedral Model in a Nutshell

Affine program regions:

- **Loops have affine control only (may over-approximate otherwise)**

Benabderrahmane, Mohamed-Walid, Louis-Noël Pouchet, Albert Cohen, and Cédric Bastoul.
"The polyhedral model is more widely applicable than you think." In *International Conference on Compiler Construction*, 2010.

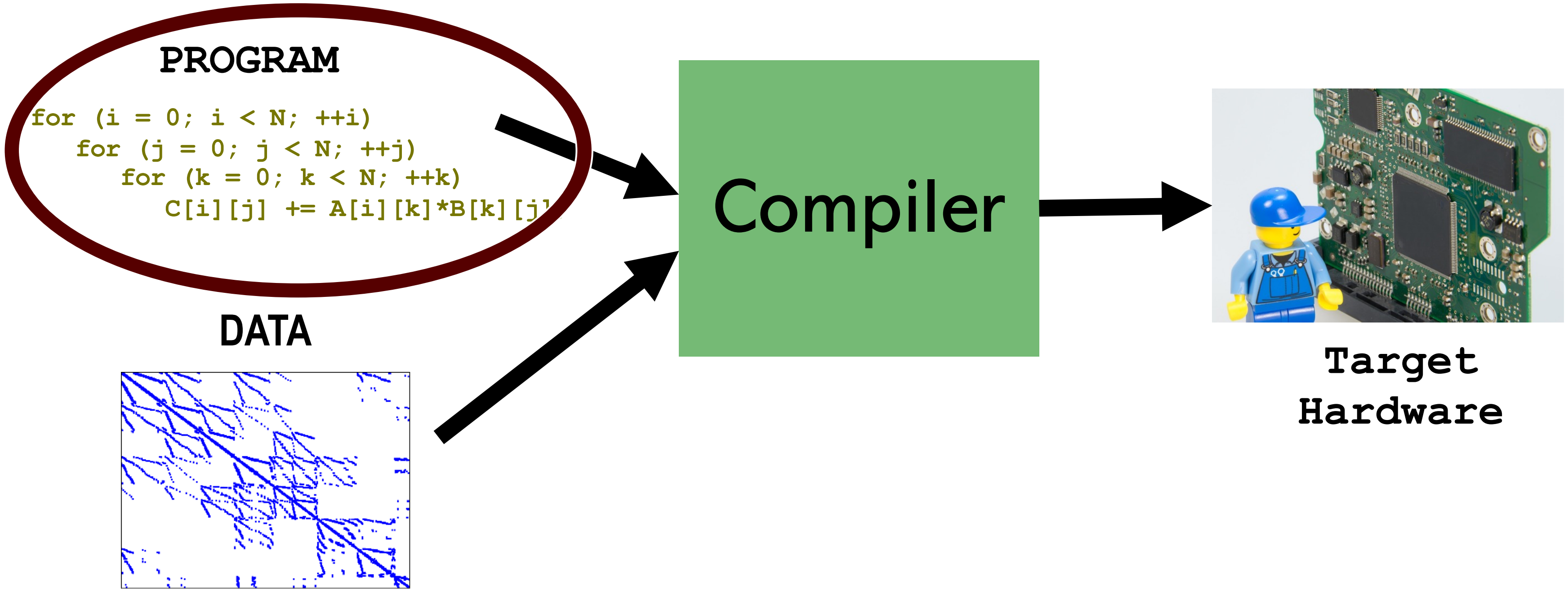
- **Iteration domain: represented as union of integer polyhedral and lattices (always possible, but may be inefficient)**

Rodríguez, Gabriel, and Louis-Noël Pouchet. "Polyhedral modeling of immutable sparse matrices."
In *8th International Workshop on Polyhedral Compilation Techniques. Manchester, UK*. 2018.

- **Array index functions: represented as functions of the loop iterators and parameters**
- **Data dependence between S1 and S2: a subset of the Cartesian product of *D*S1 and *D*S2 (exact analysis possible)**

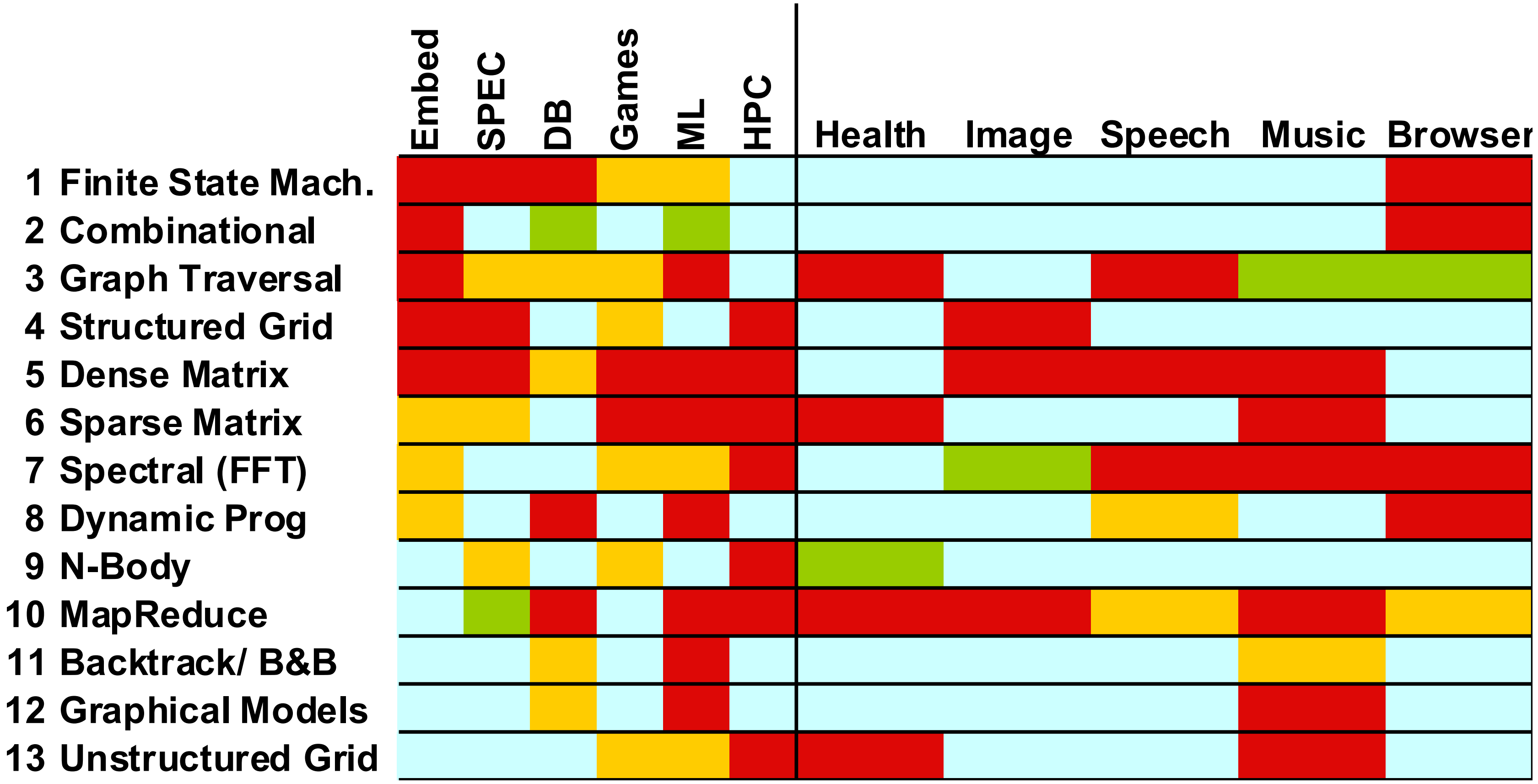
- **Precise dataflow analysis [Feautrier,88]**
- **Efficient algorithms for data locality [Bondhugula,08]**
- **Effective code generation [Bastoul,04]**
- **Computationally expensive algorithms (ILP/PIP)**

Outline: Modeling PROGRAMs



```
for (i = 0; i < N; ++i)
   for (j = 0; j < N; ++j)
      for (k = 0; k < N; ++k)
         C[i][j] += A[i][k]*B[k][j
```

Motifs In Applications (Berkeley Motifs)



	Embed	SPEC	DB	Games	ML	HPC	Health	Image	Speech	Music	Browser
1 Finite State Mach.	🟥	🟥	🟥	🟨	🟨	⬜	⬜	⬜	⬜	⬜	🟥
2 Combinational	🟥	⬜	🟩	⬜	🟩	⬜	⬜	⬜	⬜	⬜	🟥
3 Graph Traversal	🟥	🟨	🟨	🟨	🟥	⬜	🟥	⬜	🟥	🟩	🟩
4 Structured Grid	🟥	🟥	⬜	🟨	⬜	🟥	⬜	🟥	⬜	⬜	⬜
5 Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥	⬜	🟥	🟥	🟥	⬜
6 Sparse Matrix	🟨	🟨	⬜	🟥	🟥	🟥	🟥	⬜	⬜	🟥	⬜
7 Spectral (FFT)	🟨	⬜	⬜	🟨	🟨	🟥	⬜	🟩	🟥	🟥	🟥
8 Dynamic Prog	🟨	⬜	🟥	⬜	🟥	⬜	⬜	⬜	🟨	⬜	🟥
9 N-Body	⬜	🟨	⬜	🟨	⬜	🟥	🟩	⬜	⬜	⬜	⬜
10 MapReduce	⬜	🟩	🟥	⬜	🟥	🟥	🟥	🟥	🟨	🟥	🟨
11 Backtrack/ B&B	⬜	⬜	🟨	⬜	🟥	⬜	⬜	⬜	⬜	🟨	⬜
12 Graphical Models	⬜	⬜	🟨	⬜	🟥	⬜	⬜	⬜	⬜	🟥	⬜
13 Unstructured Grid	⬜	⬜	⬜	🟨	🟨	🟥	🟥	⬜	⬜	🟥	⬜

Source: J. Demmel

- Berkeley motifs identify key algorithmic paradigms, but not directly useful for code optimization
- Stencil abstraction is useful for compiler transformation: can we identify a small number of such abstractions with broad coverage?

Domain-Specific vs. Pattern-Specific Compilation

- **One pattern can occur in many domains!**
  - Simple example: stencils on dense/regular grids
  - Seen in image processing, physics simulation / PDE solving, etc.
- **All such stencils benefit from the same optimizations BUT not to the same extent!**
  - problem size, stencil shape, number of time steps, kind of convergence check, etc all depend on the domain, not the pattern
  - Ex: time-tiling, reduction optimization, etc.

Open question: is there a common optimization framework across domains?

Pattern: Static Control Flow Programs

- **In a nutshell: the set of programs whose control flow can be entirely predicted at compile-time**
- **Polyhedral framework: a subset of SCF where expressions are affine**
  - Otherwise use affine over-approximations
  - Or Union of small SCFs
- **Encapsulates with full accuracy numerous computation patterns:**
  - Stencils on dense grids
  - Dense linear algebra
  - Graph algorithms on adjacency matrix
  - Dense convolutions
  - etc.

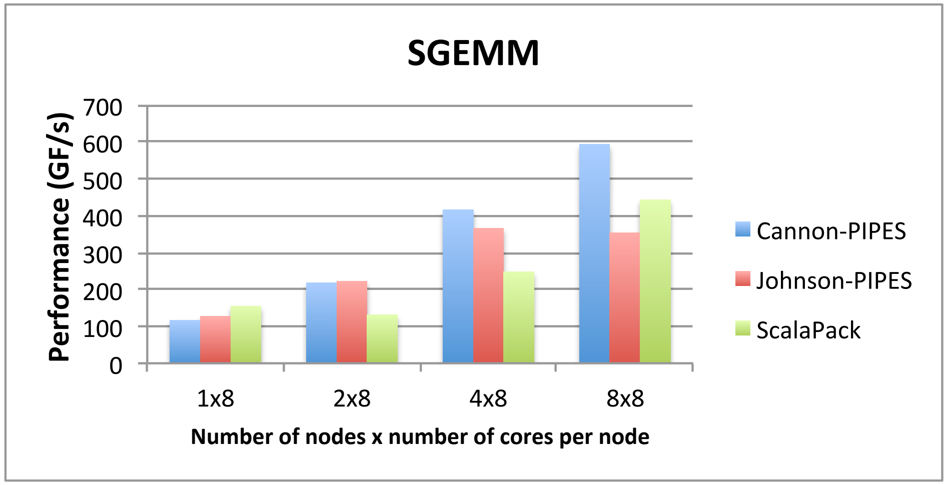
But What is Optimizing Compilation?

Main idea: ask a computer to find an equivalent program which executes faster than your own program

- Must preserve the program semantics, exploit parallel/distributed architectures, etc.
- Multiple disciplines are leveraged: algorithmic, programming, architecture, mathematics, machine learning, experimental computer science, etc.

♦ **A compelling example: programming distributed systems (PIPES)**

```
Parameter N, P;
// Define data collections
[float* A:1..N,1..N];
...
// Task prescriptions
env :: (MM:1..N,1..N,1..N);
// Input/Output:
env -> [A:1..N,1..N];
...
[C:1..N,1..N,N] -> env;
// Task dataflow
[A:i,k],[B:k,j],[C:i,j,k] -> (MM:i,j,k) -> [C:i,j,k+1];
Topology Proc = Topo2D(P,P);
// Place the N tasks (i,j,*) to Proc((i/8)%P,(j/8)%P)
(MM:i,j,1..N)@Proc((i/8)%P,(j/8)%P);
// Circular communication pattern for Cannon algorithm
[A:i,k]@(MM:i,j,k) => (MM:i,(j-1)%P,k+1);
[B:k,j]@(MM:i,j,k) => (MM:(i-1)%P,j,k+1);
```



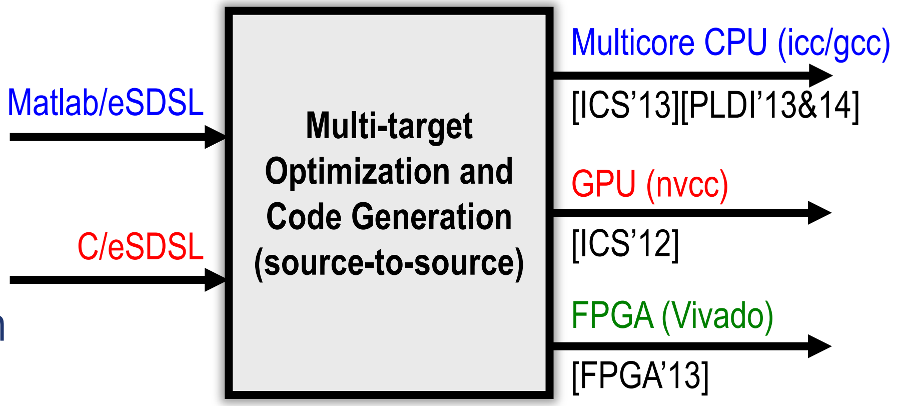
- Input: 20 lines, nearly identical to textbook, compiler generates 2000+ lines of code!

M. Kong, L.-N. Pouchet, P. Sadayappan, V. Sarkar. "PIPES: A Language and Compiler for Task-based Programming on Distributed-Memory Clusters", to appear in IEEE/ACM Supercomputing (SC'16), Nov. 2016.

StencilDSL: Embedded Domain-Specific Language

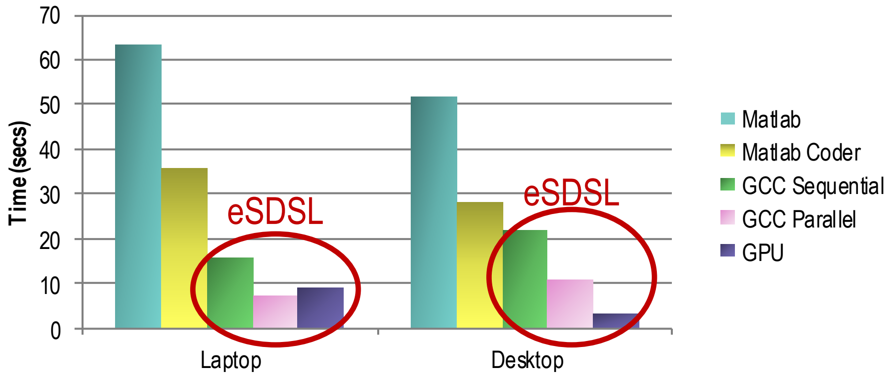
Benefits of high-level specification of computations using domain-specific languages:

- Ease of use (for mathematicians/scientists creating the code)
- Ease of optimization (facilitate loop and data transformations)
- Embedded DSL provides flexibility:
- Generality of standard programming language
- Automated transformation of embedded DSL region



```
int Nr; int Nc;
grid g [Nr][Nc];
double griddata a on g at 0,1;

pointfunction five_point_avg(p) {
  double ONE_FIFTH = 0.2;
  [1]p[0][0] = ONE_FIFTH*([0]p[-1][0] + [0]p[0][-1]
                + [0]p[0][0] + [0]p[0][1] + [0]p[1][0]);
}
iterate 1000 {
  stencil jacobi_2d {
    [0     ][0:Nc-1] : [1]a[0][0] = [0]a[0][0];
    [Nr-1  ][0:Nc-1] : [1]a[0][0] = [0]a[0][0];
    [0:Nr-1][0     ] : [1]a[0][0] = [0]a[0][0];
    [0:Nr-1][Nc-1  ] : [1]a[0][0] = [0]a[0][0];
    [1:Nr-2][1:Nc-2] : five_point_avg(a);
  }
  reduction max_diff max {
    [0:Nr-1][0:Nc-1] : fabs([1]a[0][0] - [0]a[0][0]);
  }
} check (max_diff < .00001) every 4 iterations
```

Matlab/eSDSL: Rician denoise



Stencil DSL Example

```
int Nr; int Nc;
grid g [Nr][Nc];

double griddata a on g at 0,1;

pointfunction five_point_avg(p) {
  double ONE_FIFTH = 0.2;
  [1]p[0][0] = ONE_FIFTH*([0]p[-1][0] + [0]p[0][-1]
               + [0]p[0][0] + [0]p[0][1] + [0]p[1][0]);
}

iterate 1000 {
  stencil jacobi_2d {
    [0      ][0:Nc-1] : [1]a[0][0] = [0]a[0][0];
    [Nr-1   ][0:Nc-1] : [1]a[0][0] = [0]a[0][0];
    [0:Nr-1][0      ] : [1]a[0][0] = [0]a[0][0];
    [0:Nr-1][Nc-1   ] : [1]a[0][0] = [0]a[0][0];
    [1:Nr-2][1:Nc-2] : five_point_avg(a);
  }
  reduction max_diff max {
    [0:Nr-1][0:Nc-1] : fabs([1]a[0][0] - [0]a[0][0]);
  }
} check (max_diff < .00001) every 4 iterations
```

Reference data over two time steps: current(0) and next (1)

Time Loop

Boundary

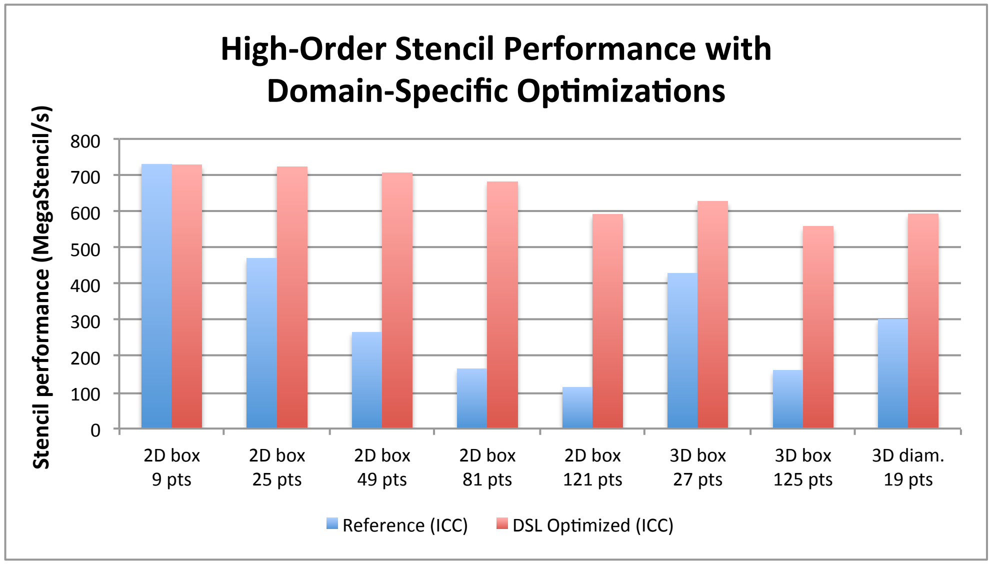
Interior

Optimizations for High-Order Stencils

- **Significance and Impact:**
  - High-order stencils arise in high-accuracy methods for PDEs in various domains
  - Prior view: the higher the discretization order, the lower the throughput
  - With proposed optimization system: maintains throughput when using higher accuracy methods!
- **DSL Technologies for Exascale Computing**
  - DoE XStack, w/ Dan Quinlan (LLNL)
  - Chombo: high-order methods
  - PolyOpt: key technology in DTEC
  - **Reusable PSL optimizer across a range of DSLs**

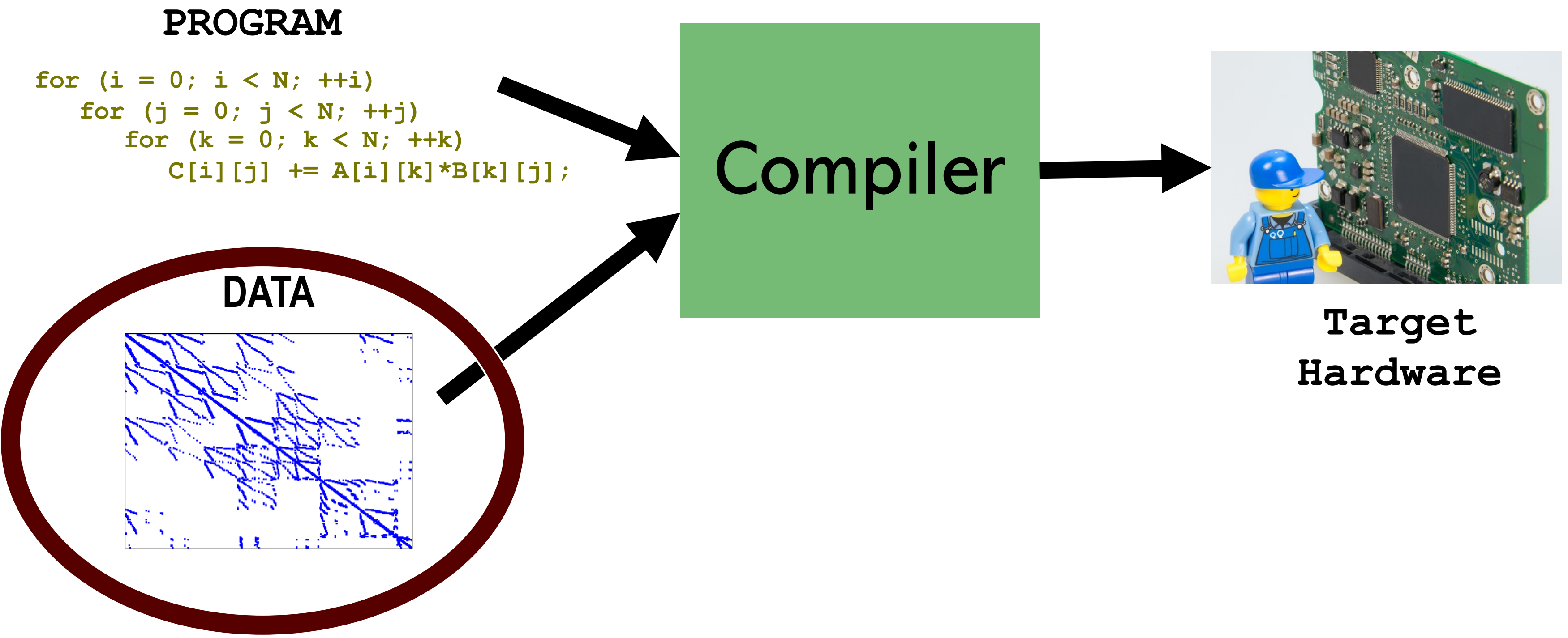
D-TEC

Techniques for Building Domain Specific Languages (DSLs)



K. Stock, M. Kong, L.N. Pouchet, T. Grosser, F. Rastello, J. Ramanujam, and P. Sadayappan, **"A Framework for Enhancing Data Reuse via Associative Reordering" (PLDI 2014)**
Available in ROSE/D-TEC branch.

Outline: Modeling DATA



```
for (i = 0; i < N; ++i)
    for (j = 0; j < N; ++j)
        for (k = 0; k < N; ++k)
            C[i][j] += A[i][k]*B[k][j];
```

Application: Implement On-Chip Data Reuse

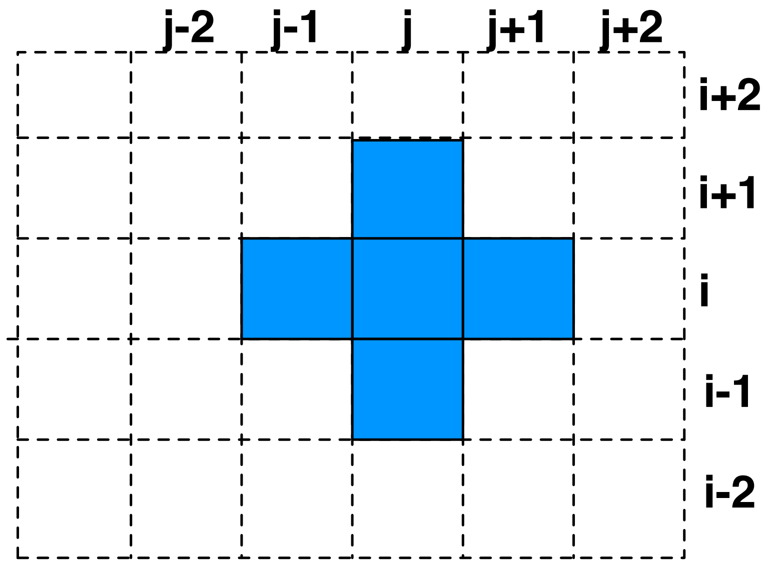
- **Key ideas:**
  - Compute the set of data used at a given loop iteration
  - Reuse data between consecutive loop iterations
  - Process works for any loop in the program
  - Natural complement of tiling: the tile size will determine how much data is read by a non-inner-loop iteration

The polyhedral framework can be used to easily compute all this information, including what to communicate

  - Address generation / data preloading functions / prefetching
  - Accelerator communication code (copy-in/copy-out)
  - …

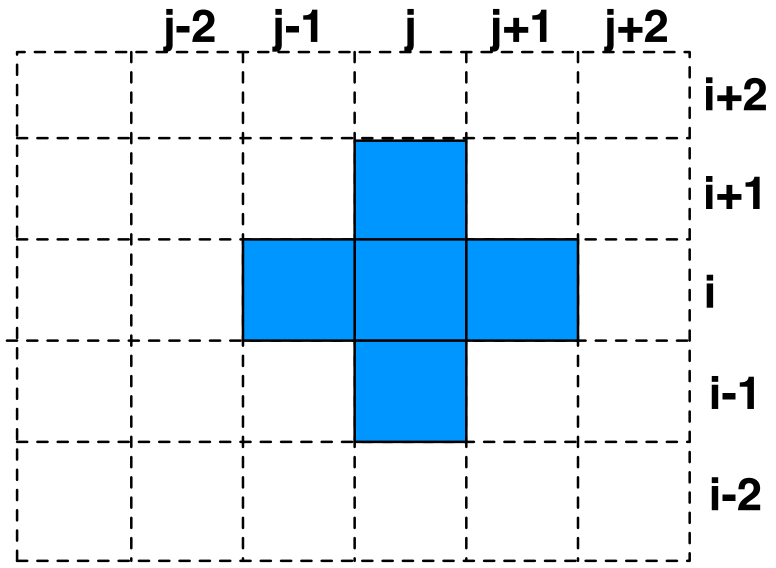
Pouchet, Louis-Noel, Peng Zhang, Ponnuswamy Sadayappan, and Jason Cong. "Polyhedral-based data reuse optimization for configurable computing." In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013.

Computing the Per-iteration Data Reuse



```
// Two-dimensional Jacobi-like stencil
for (t = 0; t < T; ++t)
  for (i = 0; i < N; ++i)
    for (j = 0; j < N; ++j)
       B[i][j] = 0.2*(  A[i][j-1]
                      + A[i][j]
                      + A[i][j+1]
                      + A[i-1][j]
                      + A[i+1][j]);
```

Computing the Per-iteration Data Reuse

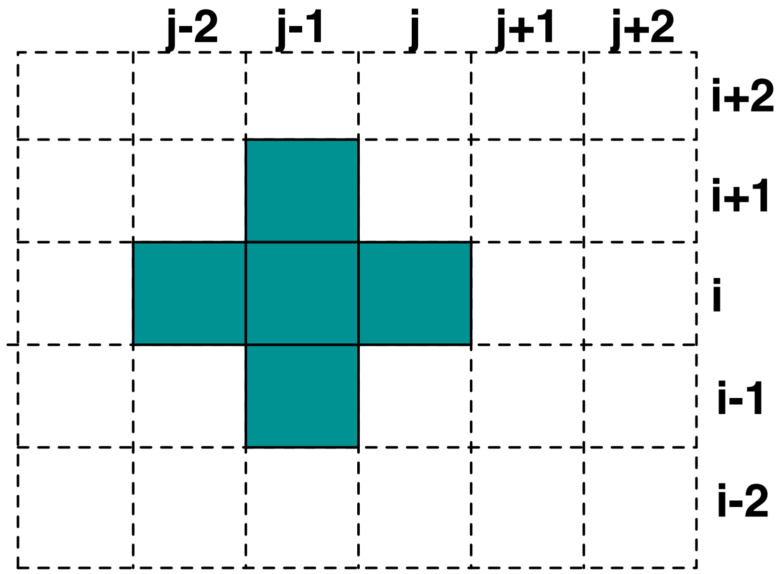


Compute the data space of A, at iteration $\vec{x} = (t, i, j)$

$$DS_A(\vec{x}) = \bigcup_{s \in S} FS_A^s(\vec{x})$$

$F(\vec{x})$ is the image of $\vec{x}$ by the function F.

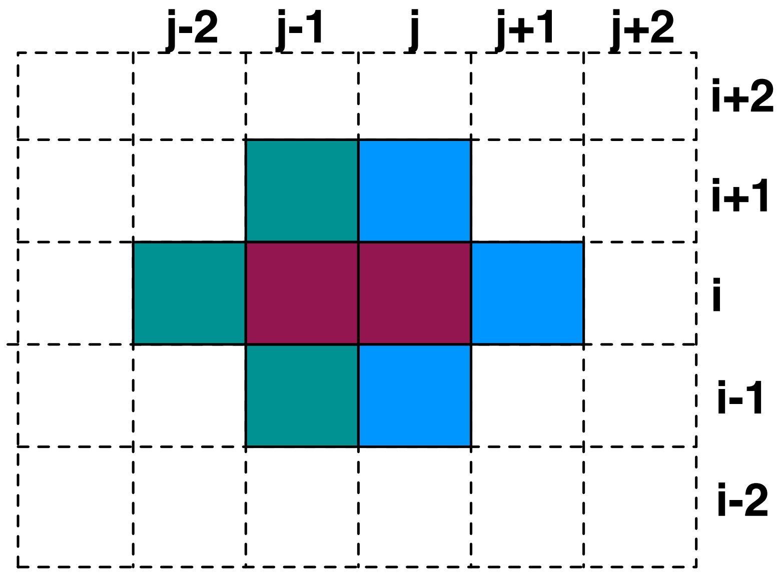
Computing the Per-iteration Data Reuse



Compute the data space of A, at iteration $\vec{y} = (t, i, j-1)$

$$DS_A(\vec{y}) = \bigcup_{s \in S} FS_A^s(\vec{y})$$

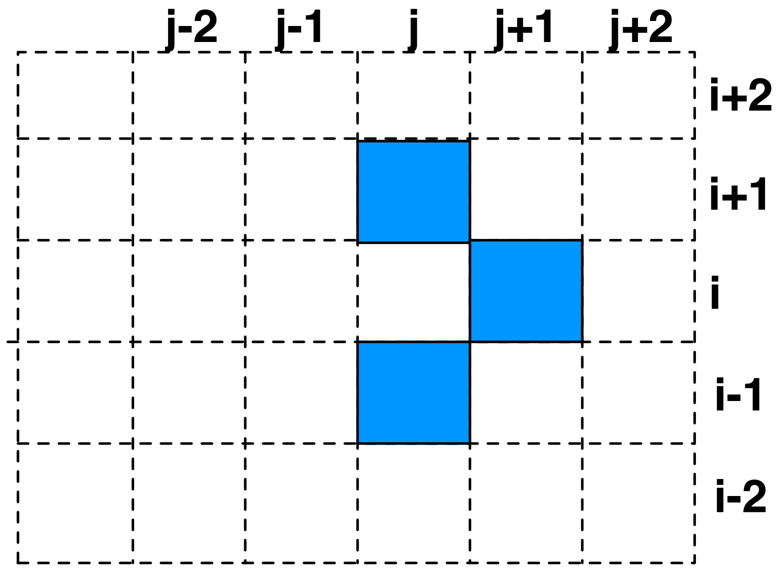
Computing the Per-iteration Data Reuse



Reused data: red set

$$ReuseSet = DS_A(\vec{x}) \cap DS_A(\vec{y})$$

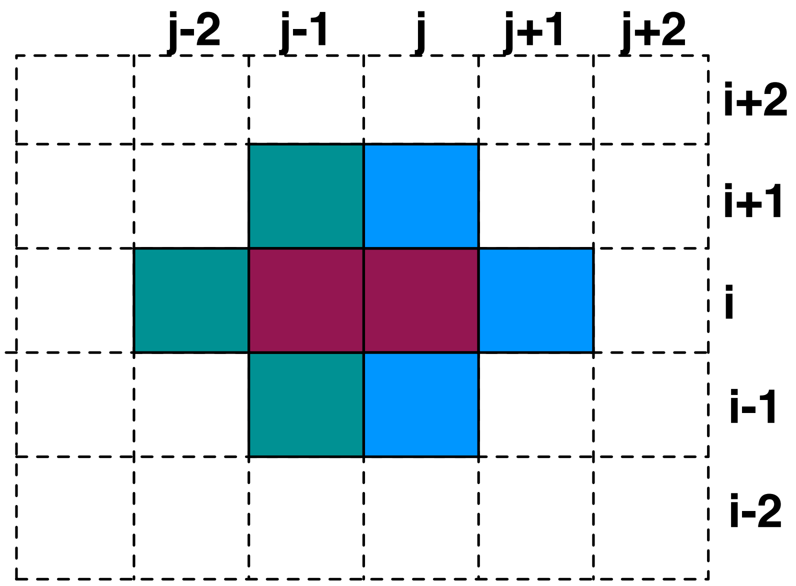
Computing the Per-iteration Data Reuse



Per-iteration communication: blue set

$$PerCommSet = DS_B(\vec{x}) - ReuseSet$$

Computing the Per-iteration Data Reuse



These sets are parametric polyhedral sets

- Use CLooG to scan them
- Work **for any value of t,i,j**

$\rightarrow$ **an initialization copy is executed before the first iteration of the loop, and communications are done at each iteration**

Pouchet, Louis-Noel, Peng Zhang, Ponnuswamy Sadayappan, and Jason Cong. "Polyhedral-based data reuse optimization for configurable computing." In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013.

Parametric Slicing: Easy Manipulation of Sub-Spaces

DEFINITION 2 (PARAMETRIC DOMAIN SLICE). *Given a loop nest with a loop l of depth n surrounded by $k-1$ loops, and an integer constant α, the parametric domain slice (PDS) of loop l is a subset of $\mathbb{Z}^n$ defined as follows:*

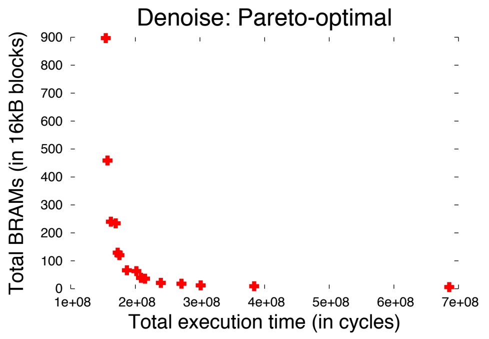
$$P_{l,\alpha} = \{(x_1,\ldots,x_n) \in \mathbb{Z}^n | x_1 = p_1,\ldots,x_{k-1} = p_{k-1}, x_k = p_k + \alpha\}$$

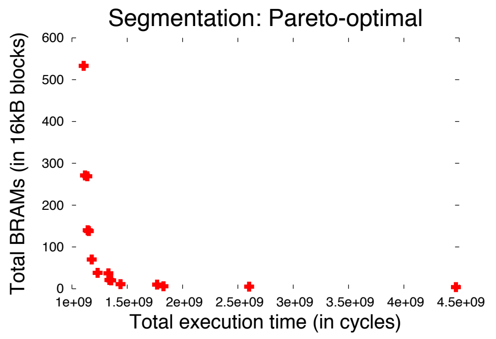
where $p_1,\ldots,p_n$ are parametric constants unrestricted on $\mathbb{Z}$.

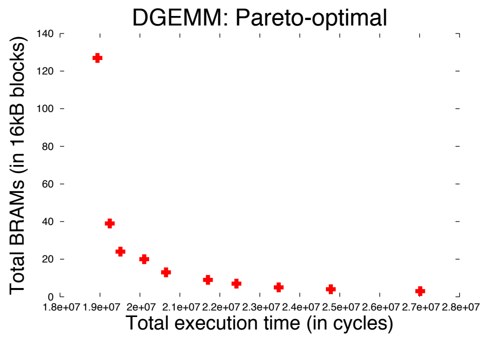
```
PDS_TIJ := [T,I,J,TN,N] -> { [t,i,j] : t = T and i = I and j = J };
PDS_TIJm1 := [T,I,J,TN,N] -> { [t,i,j] : t = T and i = I and j = J-1 };
DS := [T,I,J,TN,N] -> { [t,i,j] : 0 <= t < TN and 1 <= i,j < N - 1 };
FA := [T,I,J,TN,N] -> { [t,i,j] -> A[i-1][j]; [t,i,j] -> A[i][j]; [t,i,j] ->
       A[i+1][j]; [t,i,j] -> A[i][j-1]; [t,i,j] -> A[i][j+1] };

curiter := FA(DS * PDS_TIJ);
previter := FA(DS * PDS_TIJm1);
reuseset := curiter * previter;
card reuseset;
codegen reuseset;
```

- **Easily represent data space touched by e.g. one iteration of loop i**
- **Polyhedral/Presburger set: can be code-generated by ISL, can be counted, etc.**

Experimental Results







Benchmark	Description	basic off-chip	PolyOpt	hand-tuned [17]
denoise	3D Jacobi+Seidel-like 7-point stencils	0.02 GF/s	4.58 GF/s	52.0 GF/s
segmentation	3D Jacobi-like 7-point stencils	0.05 GF/s	24.91 GF/s	23.39 GF/s
DGEMM	matrix-multiplication	0.04 GF/s	22.72 GF/s	N/A
GEMVER	sequence of matrix-vector	0.10 GF/s	1.07 GF/s	N/A

- Convey HC-1 (4 Xilinx Virtex-6 FPGAs), total bandwidth up to 80GB/s
- AutoESL version 2011.1, use memory/control interfaces provided by Convey
- Core design frequency: 150MHz, off-chip memory frequency: 300HMz

Data-Specific Compilation

Main idea: synthesize code that is specialized to a specific sparse data structure, using polyhedra

- **Irregular and sparse data structures are central in scientific computing and in machine learning**
  - Graph processing, neural net inference after weight pruning, etc.
- **Typical approach: encode the sparse structure in some format, and provide a generic executor code to traverse the data**
- **Proposed approach: encode the sparse structure with polyhedra, and generate a specialized executor code for that structure**
- **Tunable: target SIMD / performance, target compression / code size, etc.**
- **General: works for n-dimensional sparse data structures (e.g., sparse tensors)**

T. Augustine, J. Sharma, L.-N. Pouchet, G. Rodriguez**, "Generating Piecewise-regular code from Irregular Structures" (PLDI 2019)**

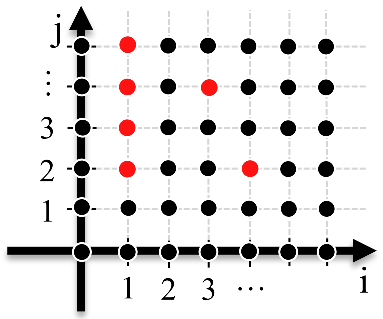
Computing on Sparse Structures

Compressed Sparse Row (CSR) code for sparse matrix vector multiply

```
for (i = 0; i < nrows; i++)
    for (j = pos[i]; j <= pos[i+1]; j++)
        y[i] += csr_data[j] * x[cols[j]];
```

- Code is generic for any sparse matrix
- For every nonzero of the matrix, performs 4 memory reads
- SIMD vectorization requires gather/scatter, code is not regular/polyhedral

Code specialized for one specific sparsity structure:



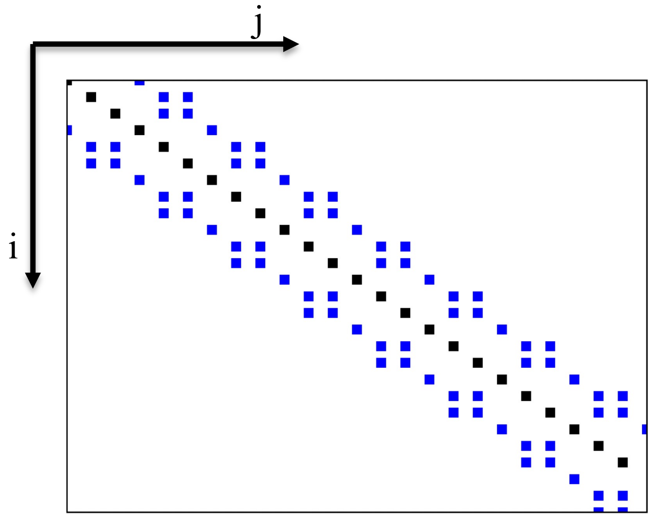
```
for (j = 2; j <= 5; j++)
   y[1] += csr_data[j-2] * x[j];
y[3] += csr_data[5] * x[4];
y[4] += csr_data[6] * x[2];
```

T. Augustine, J. Sharma, L.-N. Pouchet, G. Rodriguez, **"Generating Piecewise-regular code from Irregular Structures" (PLDI 2019)**

And What is a Sparse Structure?

Here, a sparse structure is simply a series of integer tuples

Example: a sparse matrix is represented by the tuple (i,j,data)

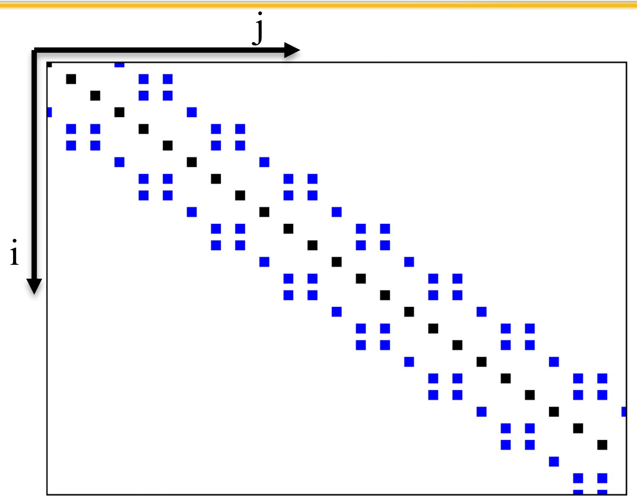


HB/nos1 matrix from SuiteSparse

	i	cols[j]	&(A_data[j])
1:	0	0	0x00
2:	0	3	0x04
3:	1	1	0x08
4:	1	4	0x0C
5:	1	5	0x10
6:	2	2	0x14
7:	2	4	0x18
8:	2	5	0x1C
9:	3	0	0x20
10:	3	3	0x24
11:	3	6	0x28
…	…	…	…

We handle sparse structures of arbitrary dimensionality, this includes sparse tensors

Encoding Sparsity with Polyhedra



HB/Nos1 matrix from SuiteSparse

	i	cols[j]	&(A_data[j])
1:	0	0	0x00
2:	0	3	0x04
3:	1	1	0x08
4:	1	4	0x0C
5:	1	5	0x10
6:	2	2	0x14
7:	2	4	0x18
8:	2	5	0x1C
9:	3	0	0x20
10:	3	3	0x24
11:	3	6	0x28

D1 : { [i,j,k] : i = 2 and 4 <= j <= 5 and k = 4j + 8 }

D2: { [i,j,k] : 2 <= i <= 3 and i = j and k = 16i – 12 }

When modeling problems like SpMV, we consider the trace reorderable

That is, non-consecutive points in the original trace may be grouped together

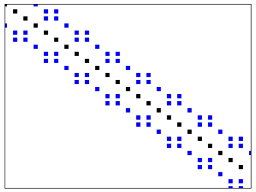
Representing Integer Tuples as Z-Polyhedra

- **A Z-Polyhedron models sets of integer tuples, with “holes”**
- **A sparse structure is a list of integer tuples, or points**
- **So we can represent a sparse structure as a union of Z-polyhedra!**
  - Target scenario: many points can be captured in a single polyhedron
  - Performance objective: polyhedra should be easy to SIMD vectorize

- **Challenges:**
  1. How to determine the shapes (polyhedron and lattice) that captures the largest number of points, *efficiently*?
  2. How to reach good performance for e.g. SpMV programs encoded as polyhedra?

Complexity Trade-Off [1/2]

- **A Z-Polyhedron may use more dimensions than the tuple size**
  - Think tiling a 2D iteration space: you obtain a new 4D iteration space, but that still describes exactly the same original set of 2D points

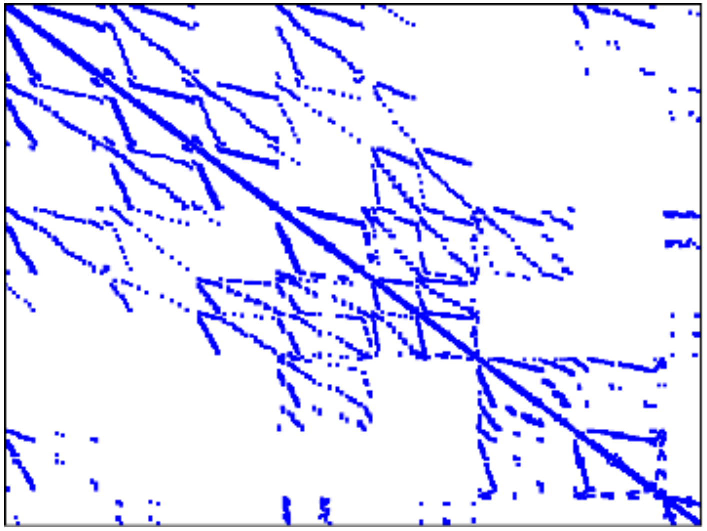


max_d	2	3	4	5	6	7	8
pieces	312	159	81	4	3	2	1
cycles	11373	11583	9938	35730	34116	39306	50371
LoC	772	1004	671	195	368	165	101

- **Using more variables/dimensions in the polyhedron (maxd) reduces the number of polyhedra needed (pieces) to capture the full matrix**
  - Leads to better compaction (LoC)
- **<u>But it does not necessarily lead to better performance</u>**

Complexity Trade-Offs [2/2]

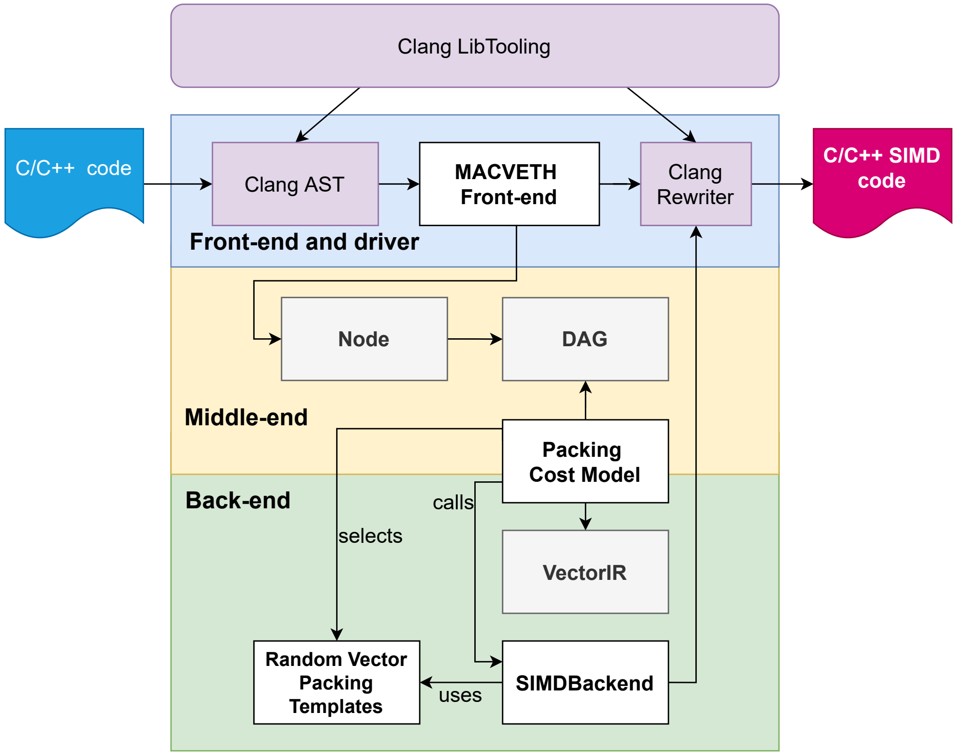
- **Complex sparse structures need many polyhedra to capture them**
  - This sparse matrix, HB/can_1072 is reconstructed with 870 polyhedra, of up to 8 dimensions
  - Code size is directly related to the number of polyhedra needed



- **In this work, we design a series of algorithms that trade-off the number of polyhedra needed versus their "complexity"**
  - Try simple shape first: "rectangles", with regular strides (SIMD-friendly)
  - Try more complex shapes afterwards (skewed ones, with many dimensions)

MACVETH: Automatic SIMD Vectorization for Codelets

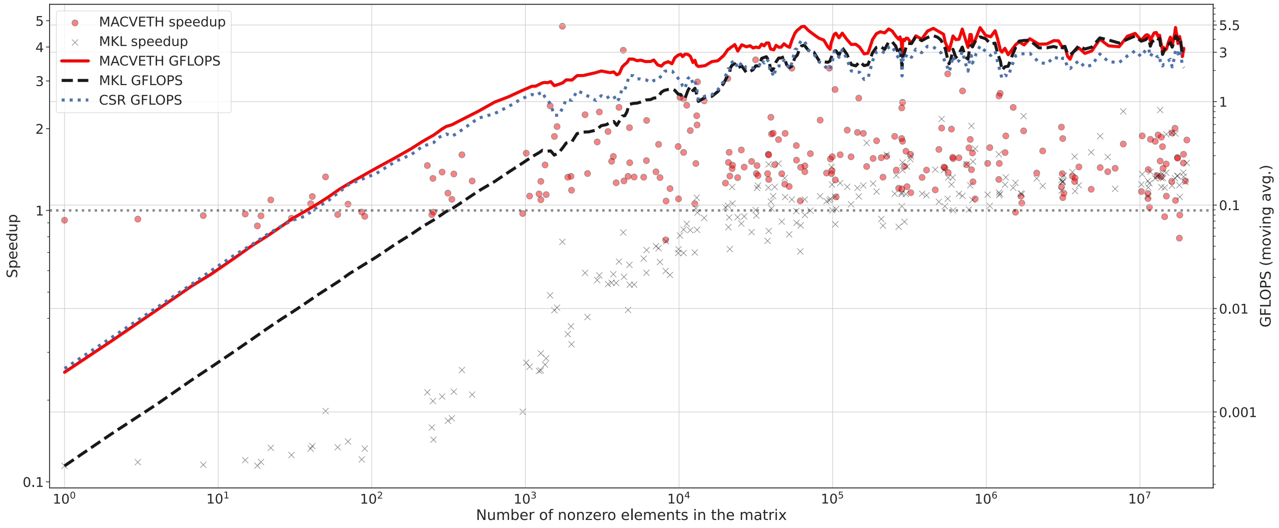
```
void kernel_spmv_fragment_0(float *__restrict A,
                            float *__restrict x,
                            float *__restrict y)
   register int i0;
   for (i0 = 0; i0 <= 1; ++i0)
     y[1] += A[i0] * x[i0];
   for (i0 = 0; i0 <= 2; ++i0)
     y[2] += A[i0 + 2] * x[i0];
   for (i0 = 0; i0 <= 1; ++i0)
     y[3] += A[i0 + 5] * x[i0 + 1];
   for (i0 = 0; i0 <= 1; ++i0)
     y[4] += A[i0 + 7] * x[i0 + 1];
   y[5] += A[9] * x[1];
   for (i0 = 0; i0 <= 1; ++i0)
     y[5] += A[i0 + 10] * x[2]; }
```



```
__vop2 = _mm256_loadu_ps(&z[0]);
__vop0 = _mm256_hadd_ps(__vop0, __vop2);
__mv_lo128 = _mm256_castps256_ps128(__vop0);
__mv_hi128 = _mm256_extractf128_ps(__vop0, 0x1);
__mv_lo128 = _mm_add_ps(__mv_lo128, __mv_hi128);
__mv_hi128 = _mm_shuffle_ps(__mv_lo128, __mv_lo128,
                            0b00110001);
__mv_lo128 = _mm_add_ps(__mv_lo128, __mv_hi128);
tmp0 = tmp0 + __mv_lo128[0];
tmp1 = tmp1 + __mv_lo128[2];
```

- Extensively profile the machine to get best way to pack randomly placed data into SIMD vectors
- Use synthesis for specific SIMD packing recipe (SMT solver for masks, etc.)
- At compile-time, use the recipes with MACVETH, and pack multiple small reductions on same SIMD vector

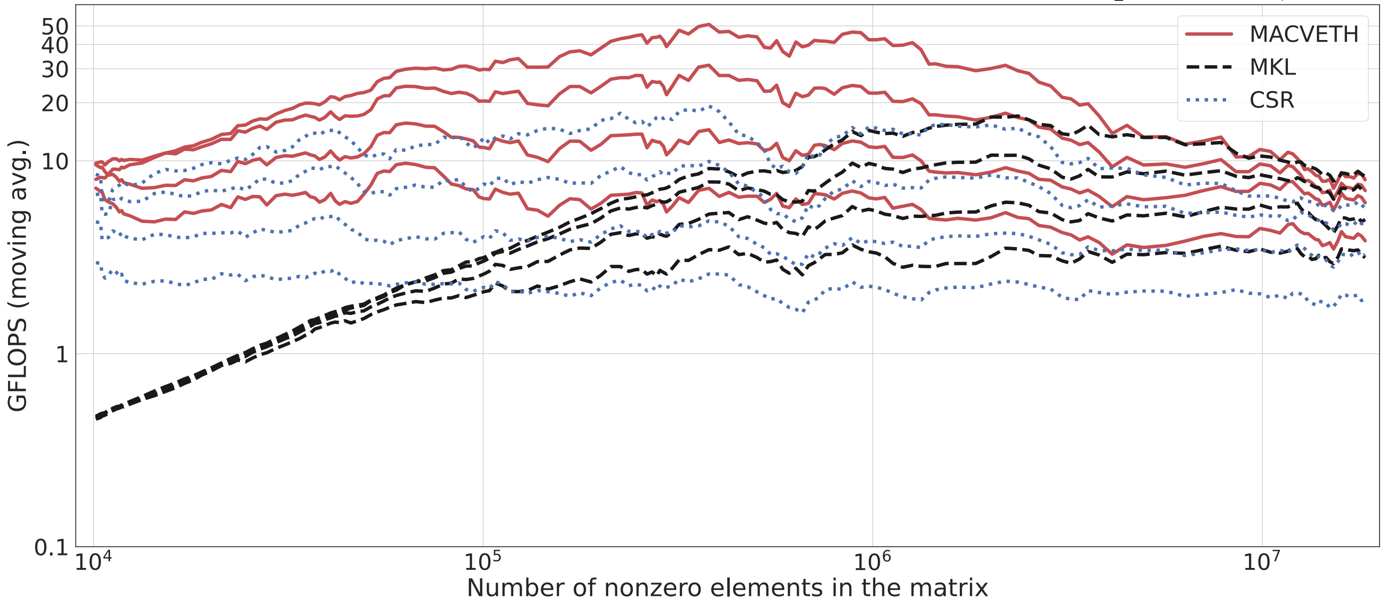
Performance Results on Intel Core i9 12900K



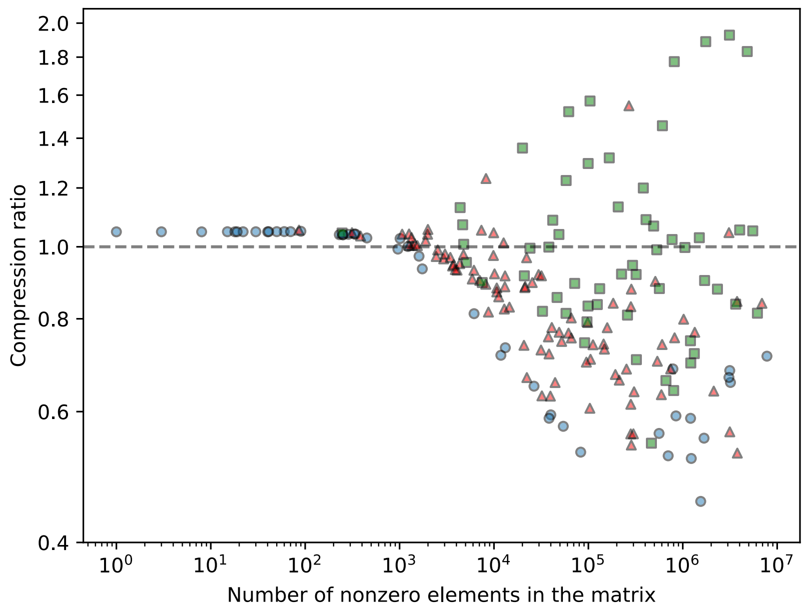
Single-core experiments

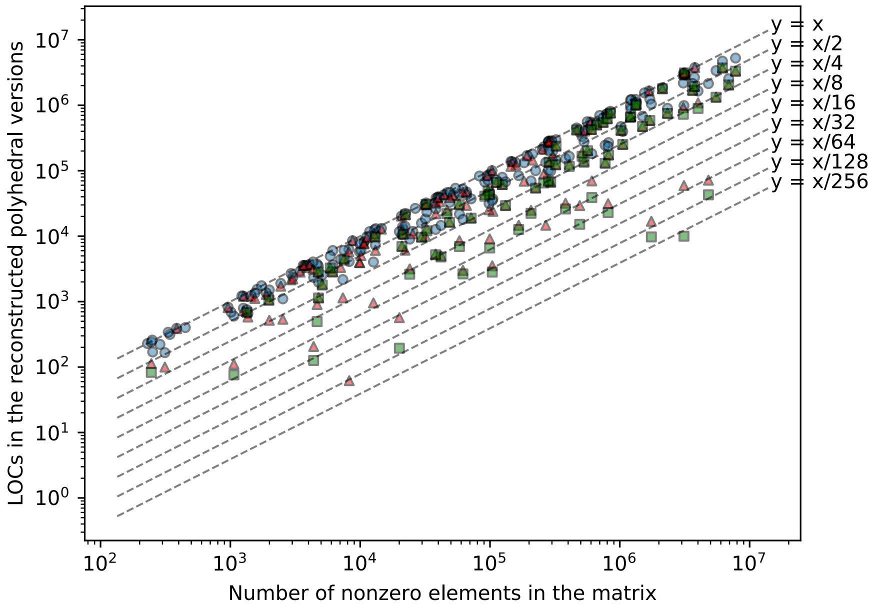
Cache	Version	Performance (GFLOPS) >1	>10K	>1M	> 10M	Peak
Cold	CSR	1.43	2.15	2.27	2.39	5.26
	DSCG	1.42	2.03	2.00	2.07	4.55
	MKL	1.15	2.56	3.07	3.29	5.31
	MACVETH	2.16	3.41	3.30	3.37	7.91
Hot	CSR	2.55	2.70	2.48	2.50	6.80
	DSCG	3.81	3.45	2.29	2.12	11.48
	MKL	3.29	3.81	3.33	3.31	7.13
	MACVETH	4.91	5.27	3.92	3.53	17.48

Multi-core experiments (hot cache)



Experimental Results: Compression



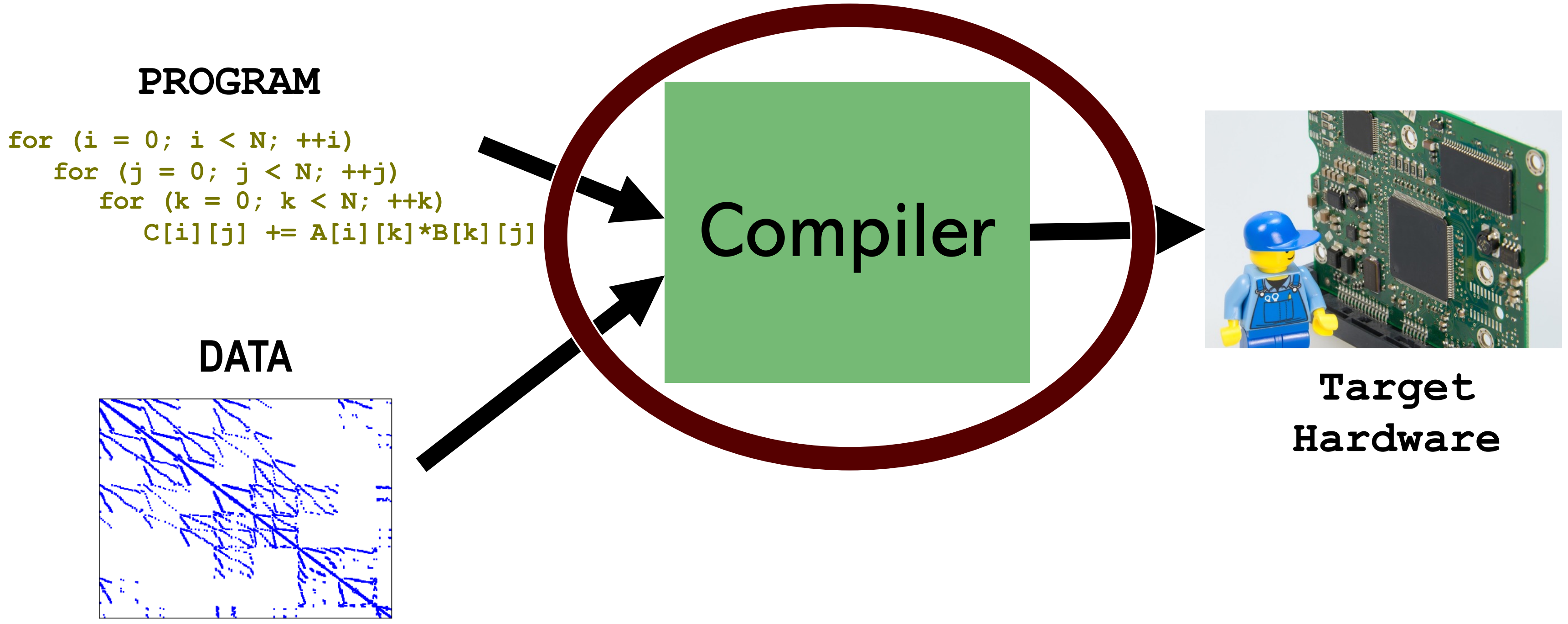


Best compression achieved
(not necessarily best performance)

Generated code size versus
number of nonzeros

- **Compression ratio: CSR footprint / size of data+code generated**
  - Best compression is achieved with different codelets, different objectives/trade-offs than for performance
  - **For better results ☺ see talk by Gabriel Rodriguez at 5pm today @ IMPACT'23!**

Outline: Scheduling and Code Optimizations



```
for (i = 0; i < N; ++i)
   for (j = 0; j < N; ++j)
      for (k = 0; k < N; ++k)
         C[i][j] += A[i][k]*B[k][j]
```

On To Optimizing Compilers

- **Focusing on loop-intensive programs**
  - Example: sequence of linear algebra operations
  - Usually, significant data reuse potential
  - Usually, significant inherent parallelism available
- **Program transformations may be required to:**
  - Exploit the data reuse (i.e., fusion and/or tiling)
  - Exploit coarse-grain parallelism (i.e., permutation)
  - Exploit fine-grain parallelism (i.e., permutation and/or distribution)
- **Key problems in mapping this software on hardware:**
  - Challenge 1: conflicting objective (locality vs. SIMD)
  - Challenge 2: different granularity (coarse-grain SMP vs. instruction selection)
  - Challenge 3: optimality (global solution vs. multi passes)

Combining High-Level and Low-Level Transformations (1)

- **High-level transformations are about program-wide restructuring**
  - Usually applied on a "large" sub-program
  - Abstract metrics should be used (e.g., "parallelization")
- **Low-level transformations are about loop/statement compilation**
  - Smaller granularity/scope of application
  - Constraints closer to the actual hardware (e.g., "aligned access")

<u>Issue: how to ensure maximal effectiveness</u>

<u>of a “local” optimization?</u>

Combining High-Level and Low-Level Transformations (2)

Main idea: Define a contract between the two compilers

- **This contract determines properties on the shape of the output code produced by the high-level transformation stage**
- **By construction code segments fitting this contract can be effectively compiled to the target hardware**
- **Example: contract for CPU SIMD synthesis using SPIRALgen**

Research problem: what is this contract?

Kong, Martin, Richard Veras, Kevin Stock, Franz Franchetti, Louis-Noël Pouchet, and Ponnuswamy Sadayappan. "When polyhedral transformations meet SIMD code generation." In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 2013.

The Contract With SPIRALgen

SPIRAL can effectively vectorize program regions of the form:

- A single, inner-most loop (requirement)
- No loop-carried dependence along this loop (requirement)
- As many instructions as possible in this loop (performance objective)
- No unaligned store (requirement)
- As few unaligned load as possible (performance objective)
- As much data reuse in the loop as possible (performance objective)
- Only stride-0 and stride-1 references (requirement)

<u>**Problem: how to restructure the code**</u>

<u>**to expose maximal candidate codelets?**</u>

Polyhedral Scheduling to The Rescue

- Use the polyhedral model to encode all requirements as constraints on the schedule of operations
- Ditto for performance objectives, encoded as optimization variables
- See paper for details (formulation is quite complex ☺)

Kong, Martin, Richard Veras, Kevin Stock, Franz Franchetti, Louis-Noël Pouchet, and Ponnuswamy Sadayappan. "When polyhedral transformations meet SIMD code generation." In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 2013.

- Other approach: specialize the scheduling strategy to the properties of the program

Kong, Martin, and Louis-Noël Pouchet. "Model-driven transformations for multi-and many-core CPUs." In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019

- Key to success: a single, convex formulation for all and only legal schedules in a certain class (implemented in PoCC/PONOS)

L.N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, R. Ramanujam, P. Sadayappan, N. Vasilache "Loop Transformations: Convexity, Pruning and Optimization" (POPL 2011)

Detour: Building Rich Transformation Spaces

- **Loop Fusion/Distribution/Code motion plays an essential role on the properties of the transformed code**
  - Trade-off data locality / communications vs. buffer size vs. parallelism
- **The number of alternatives is HUGE**
  - **Example: LU decomposition code, 11 loops >> 1,000,000,000,000 choices!**
- **Essential properties for tractability: build search spaces with:**
  - Legality: only codes which respect all data dependences
  - Uniqueness: each point in the space is a distinct transformation
  - Expressiveness: all possible compositions of transformations considered

=> LU: this space contains only 20 points!

Challenge: modeling such search spaces
to enable efficient traversal and ILP/PIP optimization

Convex Form of All, Distinct, Legal Bounded Affine Schedules

Lemma (Convex form of semantics-preserving affine schedules)

Given a set of affine schedules $\Theta^R, \Theta^S \dots$ *of dimension* m*, the program semantics is preserved if the three following conditions hold:*

(i) $\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$

(ii) $\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$

(iii) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\dots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

$$\Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) - \delta_p^{\mathcal{D}_{R,S}} + \sum_{k=1}^{p-1} \delta_k^{\mathcal{D}_{R,S}}.(K.\vec{n} + K) \geq 0$$

$\rightarrow$ Use **Farkas lemma to build all non-negative functions over a polyhedron** (here, the dependence polyhedra) [Feautrier,92]

$\rightarrow$ Bounded coefficients required [Vasilache,07]

- **Efficient ILP formulation, models strong dependence satisfaction a la Feautrier**

L.N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, R. Ramanujam, P. Sadayappan, N. Vasilache
"Loop Transformations: Convexity, Pruning and Optimization" (POPL 2011)

Convex Space of All Distinct Total Preorders

- **Fusion/distribution/code motion ⇔ total preorder**
- **Modeling: 3 binary variables per distinct pairs**
  - Example: {A}, {B}, {C}. {A,B},{C} is a TP, so is {C},{A,B}
  - $p_{A,B}$ = 1 iff A is before B, 0 otherwise
  - $e_{A,B}$ = 1 iff A is in the same class as B, 0 otherwise
  - $s_{A,B}$ = 1 iff A is after B, 0 otherwise
  - Example: {A,B},{C} is modeled as

$e_{A,B}$=1, $p_{A,C}$=1, $p_{B,C}$=1 (all others = 0)

$$O = \left\{ \begin{array}{c} 0 \le p_{i,j} \le 1 \\ 0 \le e_{i,j} \le 1 \\ 0 \le s_{i,j} \le 1 \end{array} \right\}$$

constrained to:

$$O = \left\{ \begin{array}{lrl} & \left.\begin{array}{r} 0 \le p_{i,j} \le 1 \\ 0 \le e_{i,j} \le 1 \end{array}\right\} & \text{Variables are binary} \\ & \left. p_{i,j} + e_{i,j} \le 1 \right\} & \text{Relaxed mutual exclusion} \\ \forall k \in]j,n] & \left.\begin{array}{r} e_{i,j} + e_{i,k} \le 1 + e_{j,k} \\ e_{i,j} + e_{j,k} \le 1 + e_{i,k} \end{array}\right\} & \text{Basic transitivity on } e \\ \forall k \in]i,j[& \left. p_{i,k} + p_{k,j} \le 1 + p_{i,j} \right\} & \text{Basic transitivity on } p \\ \begin{array}{l} \forall k \in]j,n] \\ \\ \forall k \in]i,j[\end{array} & \left.\begin{array}{r} e_{i,j} + p_{i,k} \le 1 + p_{j,k} \\ e_{i,j} + p_{j,k} \le 1 + p_{i,k} \\ e_{k,j} + p_{i,k} \le 1 + p_{i,j} \end{array}\right\} & \text{Complex transitivity on } p \text{ and } e \\ \forall k \in]j,n] & \left. e_{i,j} + p_{i,j} + p_{j,k} \le 1 + p_{i,k} + e_{i,k} \right\} & \text{Complex transitivity on } s \text{ and } p \end{array} \right.$$

L.N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, R. Ramanujam, P. Sadayappan, N. Vasilache
"Loop Transformations: Convexity, Pruning and Optimization" (POPL 2011)

The Convex Space of Fusions/Distributions

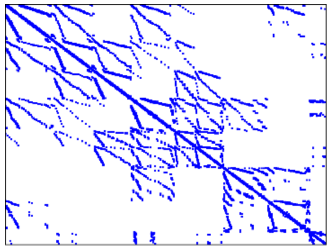
- **Starting from Total Preorders, pruning algorithm based on careful dependence analysis**
  - Exploit new properties on fusibility to accelerate the algorithm
  - Key feature: removing a class automatically removes all its superclasses
- **Numerous practical applications**
  - Find machine-specific fusion/distribution [SC'10]
  - Exploring fusion in RAJA
  - Explore function module decoupling (resource sharing) for SoCs
  - **Explore operator/kernel fusion for deep learning**
  - Tool fully automated and implemented in PoCC/PolyOpt
- **But always more to be done! ☺**
  - Build better integrated models to select fusion, more complex performance objectives
  - …

Outline: Modeling and Reasoning on the Hardware

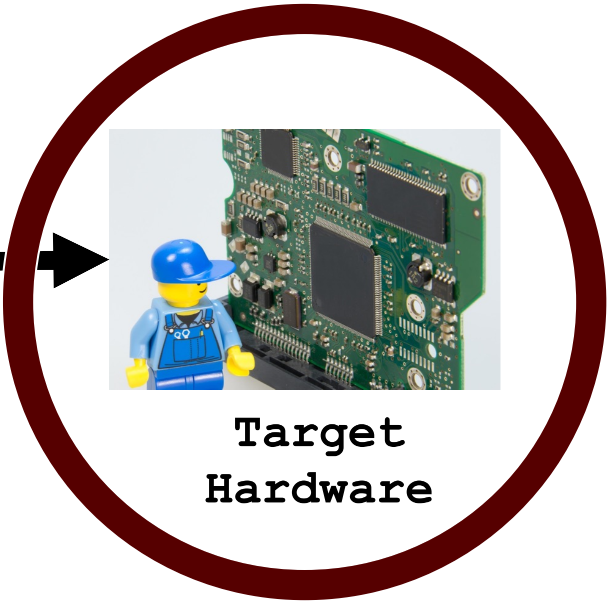
PROGRAM

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N; ++j)
        for (k = 0; k < N; ++k)
            C[i][j] += A[i][k]*B[k][j];
```

DATA



Compiler



HeteroCL: Decoupling Algorithm from Hardware Customizations

HeteroCL code

Algorithm

```
r = hcl.reduce_axis(0, 3)
c = hcl.reduce_axis(0, 3)
out = hcl.compute(N, N),
   lambda y, x:
       hcl.sum(image[x+r, y+c]*kernel[r, c],
           axis=[r, c]))
```

Declarative code (based on TVM)

Custom Compute

```
s = hcl.create_schedule()
s[out].unroll([r,c])
```

Custom Data Type

```
for i in range(2, 8):
  s.quantize([out], Fixed(i, i-2))
```

Custom Memory

```
linebuf = s[image].reuse_at(out, out.y)
winbuf = s[linebuf].reuse_at(out, out.x)
```

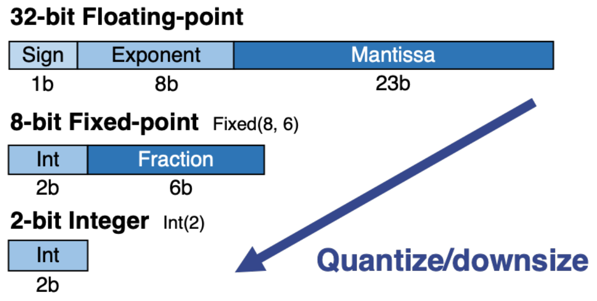
github.com/cornell-zhang/heterocl

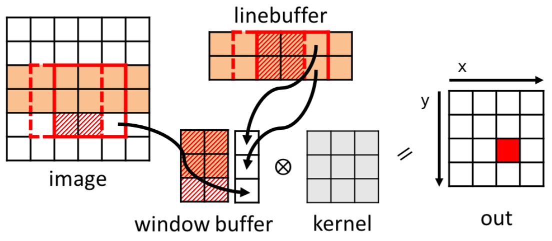
Y.-H. Lai, et al., HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing, FPGA'2019

Corresponding C code (original)

```
for (int y = 0; y < N; y++)
 for (int x = 0; x < N; x++)
  for (int r = 0; r < 3; r++)
   for (int c = 0; c < 3; c++)
    out[x, y] += image[x+r, y+c] * kernel[r, c]
```

Unroll inner loops





Source: Pr. Zhiru Zhang, Cornell

Accelerator Design with High-Level Synthesis (HLS)

Example: convolution

```
for (int y = 0; y < N; y++)
 for (int x = 0; x < N; x++)
  for (int r = 0; r < 3; r++)
   for (int c = 0; c < 3; c++)
     out[x, y] += image[x+r, y+c] * kernel[r, c]
```

Algorithm#1

Compute Customization

Algorithm#2

Data Type Customization

Memory Customization

Algorithm#3

Entangled hardware customization and algorithm

- Less portable
- Less maintainable
- Less productive

Corresponding C code (transformed)

```
#pragma HLS array_partition variable=filter dim=0
 hls::LineBuffer<3, N, ap_fixed<8,4> > buf;
 hls::Window<3, 3, ap_fixed<8,4> > window;
 for(int y = 0; y < N; y++) {
   for(int xo = 0; xo < N/M; xo++) {
#pragma HLS pipeline II=1
     for(int xi = 0; xi < M; xi++) {
       int x = xo*M + xi;
       ap_fixed<8,4> acc = 0;
       ap_fixed<8,4> in = image[y][x];
       buf.shift_up(x);
       buf.insert_top(in, x);
       window.shift_left();
       for(int r = 0; r < 2; r++)
        window.insert(buf.getval(r,x), i, 2);
       window.insert(in, 2, 2);
       if (y >= 2 && x >= 2) {
        for(int r = 0; r < 3; r++) {
         for(int c = 0; c < 3; c++) {
           acc += window.getval(r,c) * kernel[r][c];
        }}
        out[y-2][x-2] = acc;
}}}}
```

Custom compute
(Loop tiling)

Custom data type
(Quantization)

Custom memory
(Reuse buffers)

Source: Pr. Zhiru Zhang, Cornell

How To Reduce Development Cost and Increase Correctness?

Main idea: prove the transformed program is *equivalent* to the original program

- **Definition: Program Equivalence (in this talk):**

  Two programs A and B are said to be equivalent if they both compute the exact same expressions for every output memory cells written out, that is for every variable that is not local to the program.

- **If we can unambiguously determine (*decide*) that program A is equivalent to program B = transfo(A), then the transformation(s) have been correctly implemented and no bug was introduced.**

- **Alternate approach: translation validation**
  - **Keep track of the series of elementary transformations implemented, ensure the sequence preserve semantics**

Why Transformed Programs May Be Incorrect

- **You may transform the code with a correct/legal transformation, but the *tool* you use to implement the transformation *may be buggy***
  - Research compilers are often developed without rigorous testing practice
  - *Tools ok for a publication may not mean tools ready for production use*
  - Well, even production compilers are buggy!

- **The generated program after a transformation may not implement correctly that transformation**
  - Bug in the compiler, or anywhere else in the process
  - Ideally, check correctness "as late as possible" in the process, once as much transformations as feasible have been applied

- **A designer used a tool to get a first transformed program, then *manually edited* it further (typical case in HW design)**
  - Bug is introduced by the user!

Verification Tasks for HeteroCL Program Correctness

Task 1

out = F(A, B)

```
import heterocl as hcl

r = hcl.reduce_axis(0, Q)
out = hcl.compute((P, R),
      lambda x,y:hcl.sum(alpha *
           A[x,r]*B[r,y],axis=r))
s = hcl.create_schedule([A,B],gemm)
```

out' = G(A, B)

G = reorder→pipeline→unroll

```
import heterocl as hcl

r = hcl.reduce_axis(0, Q)
out = hcl.compute((P, R),
      lambda x,y:hcl.sum(alpha *
           A[x,r]*B[r,y],axis=r))
s' = hcl.create_schedule([A,B],gemm)
s'[out].reorder(out.axis[1],out.axis[0])
s'[out].pipeline(out.axis[1])
s'[out].unroll(1)
```

Does **G** model **F**?

Task 2

O_1

```
for(x = 0; x < P; ++x){
   for(y = 0; y < R; ++y) {
      out[x][y] = 0.000000e+00f;
      for(r = 0; r < Q; ++r)
         out[x][y] +=(A[x][r]*1.5)*B[r][y];
   }}
```

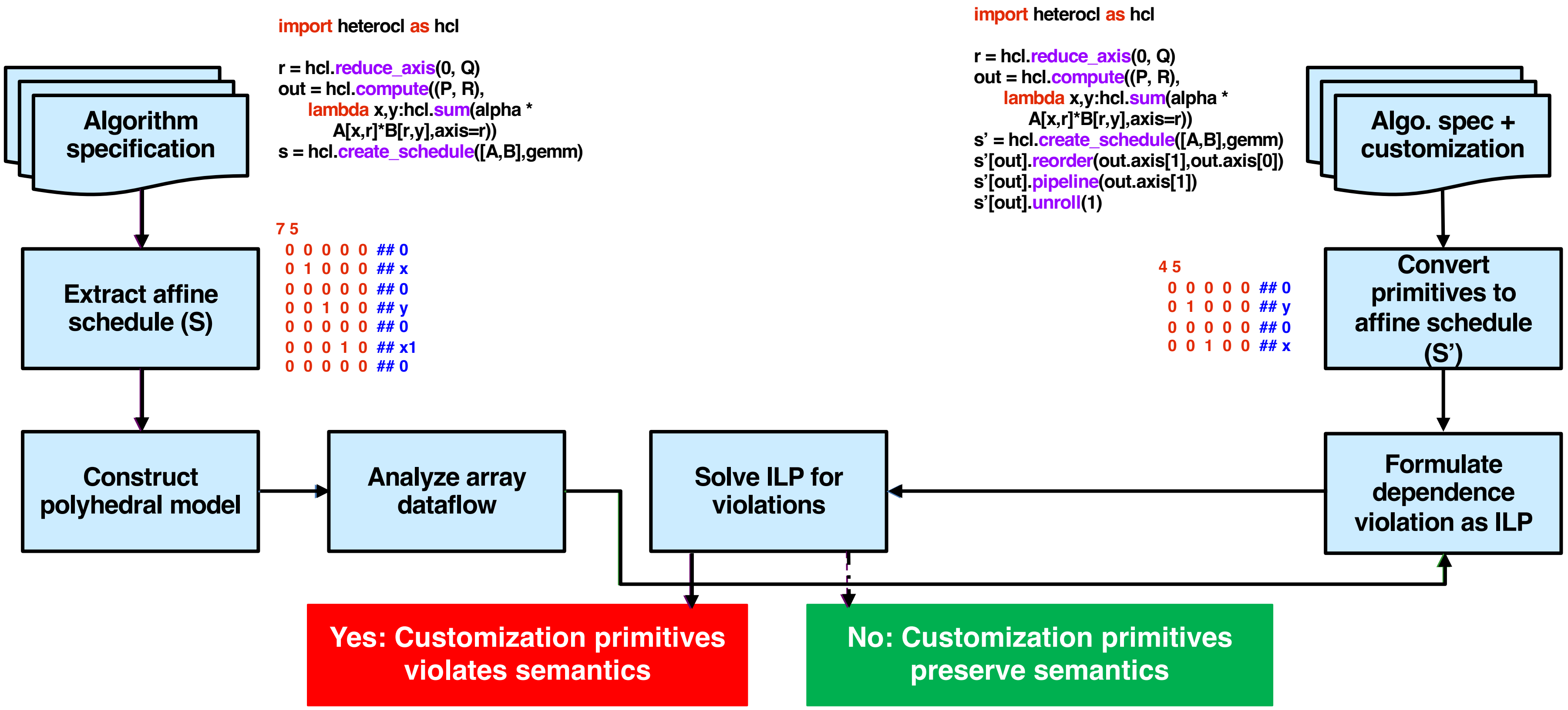
O_2

```
#pragma ii 1
for(y = 0; y < R; ++y){
   #pragma unroll
   for(x = 0; y < P; ++x) {
      for(x1 = 0; x1 < 1; ++x1)
         sum = 0.000000e+00f;
      for(r = 0; r < Q; ++r)
         sum =(A[x][r]*1.5)*B[r][y]+sum;
      out[x][y] = sum;}}
```

Is $\mathbf{O_2}$ functionally equivalent to $\mathbf{O_1}$?

Source: Dr. Debjit Pal, Cornell

Checking Compute Customizations in HCL



Algorithm specification
Extract affine schedule (S)
Construct polyhedral model
Analyze array dataflow
Solve ILP for violations
Algo. spec + customization
Convert primitives to affine schedule (S')
Formulate dependence violation as ILP
Yes: Customization primitives violates semantics
No: Customization primitives preserve semantics

```
import heterocl as hcl

r = hcl.reduce_axis(0, Q)
out = hcl.compute((P, R),
     lambda x,y:hcl.sum(alpha *
          A[x,r]*B[r,y],axis=r))
s = hcl.create_schedule([A,B],gemm)
```

```
7 5
 0 0 0 0 0 ## 0
 0 1 0 0 0 ## x
 0 0 0 0 0 ## 0
 0 0 1 0 0 ## y
 0 0 0 0 0 ## 0
 0 0 0 1 0 ## x1
 0 0 0 0 0 ## 0
```

```
import heterocl as hcl

r = hcl.reduce_axis(0, Q)
out = hcl.compute((P, R),
     lambda x,y:hcl.sum(alpha *
          A[x,r]*B[r,y],axis=r))
s' = hcl.create_schedule([A,B],gemm)
s'[out].reorder(out.axis[1],out.axis[0])
s'[out].pipeline(out.axis[1])
s'[out].unroll(1)
```

```
4 5
 0 0 0 0 0 ## 0
 0 1 0 0 0 ## y
 0 0 0 0 0 ## 0
 0 0 1 0 0 ## x
```

- Typically compute customizations are loop transformations => can be represented as change of iterations schedule, and be verified "immediately" (a few milliseconds)
- Ability to quickly display sets of valid customizations (10's to 100's to check only)
- Better approach: build the convex set of legal (affine) schedules [POPL11-PLDI19] directly

Program Equivalence with Hybrid Concrete-Abstract Interpretation (unpublished yet)

Program Equivalence is Decidable for a large class of interesting programs, under reasonable assumptions

- **Program equivalence is a fundamental problem in modern computer science**
  - **Need to ensure code transformations and optimizations are correct (environments are the same)**
  - Hardware design verification involves assessing the semantics of programs (equivalence)
  - The cost of errors can be dramatic, and errors are often silent
- **Program equivalence is more decidable than you think**
  - Combining partial evaluation, concrete interpretation of the control- and data-flow of the program, symbolic CDAG computation and tree isomorphism we can build a powerful program equivalence system
  - **Our system is limited to what it can compute**: information on loop bounds, etc. may be needed. Typically, our proof is valid for a particular problem size, the process needs to be repeated for every concrete problem sizes needed.
  - **Our system can prove equivalence irrespective of how the control/data flow is implemented, provided it is statically computable using concrete interpretation.**
- **Many "basic" equivalences, (e.g., based on valid rewrite rules), are not recognized**
  - **Possible solution by using deep learning techniques for pathfinding** (Kommrusch et al.)
  - Implement symbolic normalization of CDAGs?

Going Crazy… some example!

```
#pragma pocc-region-start liveout A,B,C,alpha,beta
    P=32*32;
    int n = P/4;
    int q = n/72+43*35-n*P;
    P = 1 + q/100000000;
    int N = 9+P; P--;
    float myqueue[2]; int queue_first = 1;
    myqueue[0] = 0;
    for (int i = 0; i < N; i++) {
      int loop_lower_bound = 42/51 *  (123/456);
      int confusing_bound = 1;
      for (int j = loop_lower_bound; confusing_bound; j++) {
          myqueue[0] = beta;
          myqueue[1] = myqueue[0];
          myqueue[0] = 0;
          C[i + 2*P][j] *= myqueue[queue_first];
          int k = 0;
          if (k >= 0)
            k+= 1;
          do {
              int tmp_jval = j + 1 + N;
              tmp_jval -= N;
              tmp_jval--;
              float tmp_mul = A[i][k-1] * B[k-1][tmp_jval];
              C[i][j] += tmp_mul * alpha;
              k++;
            }
          while (k-1 < N);
          // correct:
          confusing_bound = (j+1 < N) && confusing_bound;
          // Off-by-1:
          // confusing_bound = (j < N);
      }    }
#pragma pocc-region-end
```

?

```
#pragma pocc-region-start liveout A,B,C,alpha,beta
    int N = 10;
    for (i = 0; i < N; i++)
      for (j = 0; j < N; j++) {
         C[i][j] *= beta;
         for (k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j] * alpha;
      }
#pragma pocc-region-end
```

?

Going Crazy… some example!

```
#pragma pocc-region-start liveout A,B,C,alpha,beta
    int N = 10;
    for (i = 0; i < N; i++)
      for (j = 0; j < N; j++) {
         C[i][j] *= beta;
         for (k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j] * alpha;
      }
#pragma pocc-region-end
```

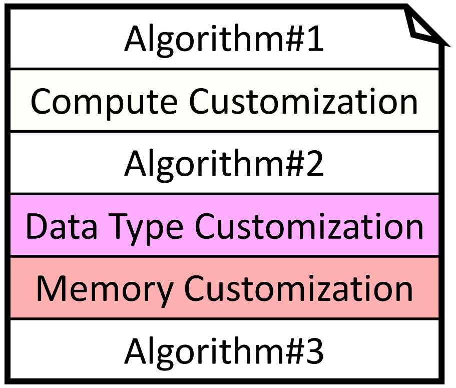
```
#pragma pocc-region-start liveout A,B,C,alpha,beta
    P=32*32;
    int n = P/4;
    int q = n/72+43*35-n*P;
    P = 1 + q/100000000;
    int N = 9+P; P--;
    float myqueue[2]; int queue_first = 1;
    myqueue[0] = 0;
    for (int i = 0; i < N; i++) {
      int loop_lower_bound = 42/51 *  (123/456);
      int confusing_bound = 1;
      for (int j = loop_lower_bound; confusing_bound; j++) {
         myqueue[0] = beta;
         myqueue[1] = myqueue[0];
         myqueue[0] = 0;
         C[i + 2*P][j] *= myqueue[queue_first];
         int k = 0;
         if (k >= 0)
           k+= 1;
         do {
             int tmp_jval = j + 1 + N;
             tmp_jval -= N;
             tmp_jval--;
             float tmp_mul = A[i][k-1] * B[k-1][tmp_jval];
             C[i][j] += tmp_mul * alpha;
             k++;
           }
         while (k-1 < N);
         // correct:
         confusing_bound = (j+1 < N) && confusing_bound;
         // Off-by-1:
         // confusing_bound = (j < N);
      }    }
#pragma pocc-region-end
```

```
$> ./bin/pocc -t --tc-orig-file dgemm-original.c --tc-trans-file dgemm-transformed.c --tc-liveout-vars "A,B,C,alpha,beta" --quiet
[PoCC] Verify equivalence of programs by abstract interpretation.
[PoCC] YES => Programs dgemm-original.c and dgemm-transformed.c are equivalent.
```

Another Example of Equivalence: Hardware Description

Example: convolution

```
N = 512;
for (int y = 0; y < N; y++)
  for (int x = 0; x < N; x++)
    for (int r = 0; r < 3; r++)
      for (int c = 0; c < 3; c++)
        out[x, y] += image[x+r, y+c] * kernel[r, c]
```



Entangled hardware customization and algorithm

- Less portable
- Less maintainable
- Less productive

Corresponding C code (transformed)

```
#pragma HLS array_partition variable=filter dim=0
 N=512; M=32;
 hls::LineBuffer<3, N, ap_fixed<8,4> > buf;
 hls::Window<3, 3, ap_fixed<8,4> > window;
 for(int y = 0; y < N; y++) {
  for(int xo = 0; xo < N/M; xo++) {
#pragma HLS pipeline II=1
    for(int xi = 0; xi < M; xi++) {
      int x = xo*M + xi;
      ap_fixed<8,4> acc = 0;
      ap_fixed<8,4> in = image[y][x];
      buf.shift_up(x);
      buf.insert_top(in, x);
      window.shift_left();
      for(int r = 0; r < 2; r++)
       window.insert(buf.getval(r,x), i, 2);
      window.insert(in, 2, 2);
      if (y >= 2 && x >= 2) {
       for(int r = 0; r < 3; r++) {
        for(int c = 0; c < 3; c++) {
          acc += window.getval(r,c) * kernel[r][c];
       }}
       out[y-2][x-2] = acc;
}}}}
```

Custom compute
(Loop tiling)

Custom data type
(Quantization)

Custom memory
(Reuse buffers)

Two programs proved equivalent!

L.-N. Pouchet and Z. Zhang. Verifying Domain-Specific Optimization in HeteroCL using Polyhedral Analysis. Intel Strategic Research Alliance, 2020-2023

Building a System for Automatic Program Equivalence

- ***When interpretation succeeds***, **our system can prove equivalence of programs (original and transformed) where:**
  - Any iteration reordering transformation (eg, loop transformations, but way beyond this class also) was applied
    - Loop permutation, loop tiling, fusion, distribution, etc.
  - "Any" control-flow implementation of the program
    - while loops, for loops, recursive calls, obfuscated induction variables, etc.
  - "Any" local storage implementation of the program
    - Scalarization, array expansion, local buffer insertion, etc.

- ***When interpretation fails, it cannot prove anything***
  - Parametric loop bounds are not handled, we typically target a tile in a tiled code. Handling parametric loop bounds may be done in some cases, e.g.:

Verdoolaege, Sven, Gerda Janssens, and Maurice Bruynooghe. "Equivalence checking of static affine programs using widening to handle recurrences." *ACM Transactions on Programming Languages and Systems (TOPLAS)* 34, no. 3 (2012)

Conclusion

- **Polyhedral Compilation is more than Affine Scheduling**
  - It is about representing programs to extract detailed semantics, but also representing data, and machine, accurately
  - Scheduling for performance remains a key problem, contributions needed!
- **Any program/data made of integer tuples can be represented as a union of polyhedra**
  - … because a single point is a polyhedron. But efficiency/usefulness is unlikely
  - Need for effective algorithms to compress these points into polyhedral
- **Hardware designs are often the result of affine transformations**
  - Can represent some hardware optimization in the polyhedral model, enabling quick verification and correctness checking
  - When the algorithm implemented is also polyhedral in nature, complex program equivalence can be proved, and properties transferred across implementations
- **There is so much more I have not covered, go read the literature now ☺**

Thank you ☺