Generating Piecewise-Regular Code from Irregular Structures

Travis Augustine[1] Jana Sarma[1] Louis-Noël Pouchet[1] Gabriel Rodríguez[2]

[1] Colorado State University, USA

[2] Universidade da Coruña, Spain

ACM SIGPLAN Conference on Programming Language Design and Implementation at 2019 ACM Federated Computing Research Conference

June 25, 2019

Funded in part by

National Science Foundation
WHERE DISCOVERIES BEGIN

UNIVERSIDADE DA CORUÑA

Colorado State University

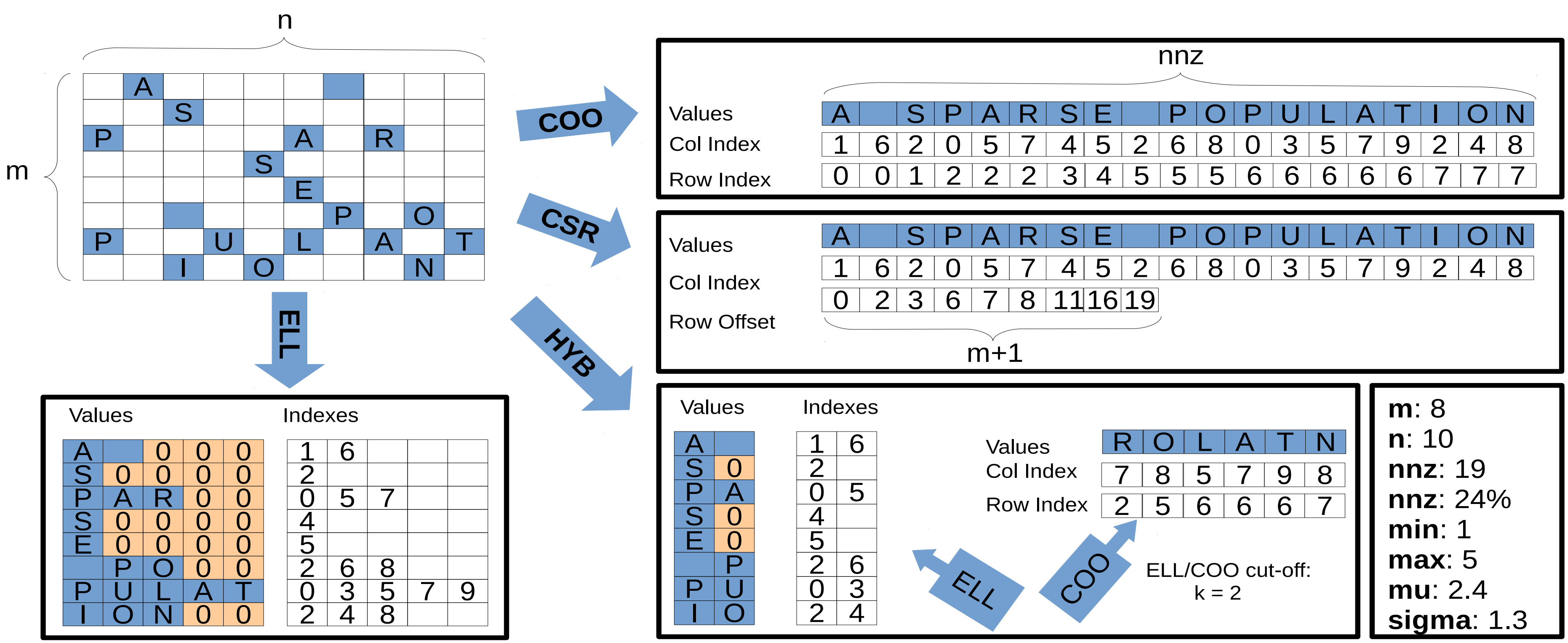
Overview

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)**

Sparse Data Representations



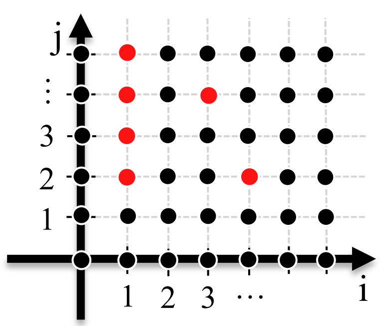
Computing on Sparse Structures

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:



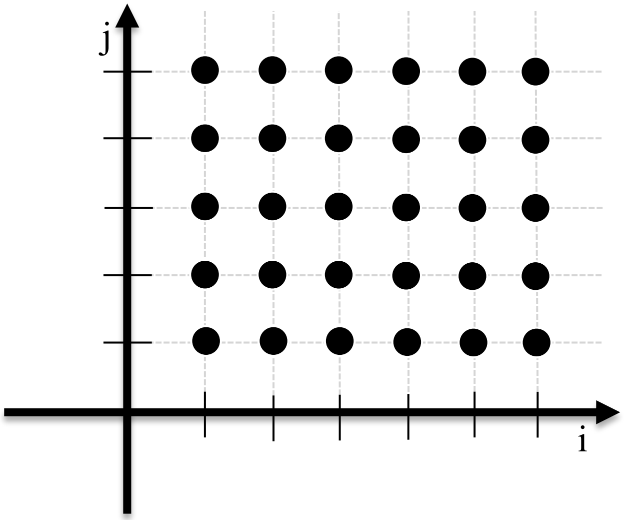

```
for (j = 2; j <= 5; j++)
   y[1] += csr_data[j-2] * x[j];
y[3] += csr_data[5] * x[4];
y[4] += csr_data[6] * x[2];
```

Application Context, Pros and Cons

- **Generating *specialized* code for one sparsity structure:**
  - Avoids the need for genericity: can remove indirection arrays / irregularity
  - Makes the loop nests easier to vectorize
  - Robust to any data changes, only the sparsity itself should not change
  - May reduce footprint, but can lead to very large code size too
  - Loses genericity: each sparse structure has a different executor program
- **Some important use cases:**
  - Sparse Matrix Vector Multiply (especially iterative SpMV)
  - Inference of some classes neural networks (especially after weight pruning)
  - Sparse tensors

But What is a Polyhedron?

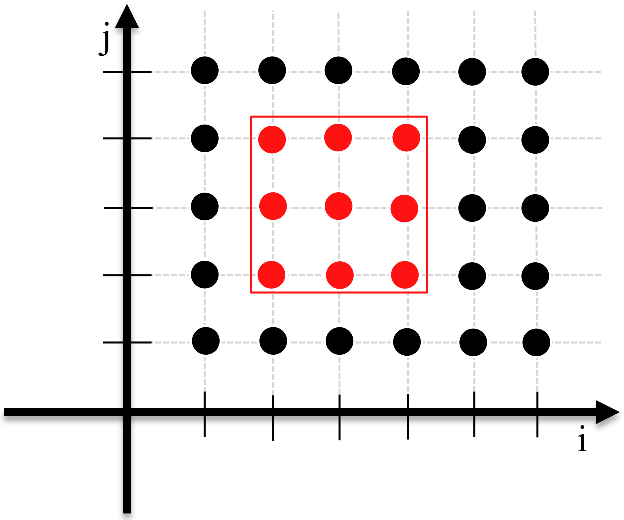
Example



Grid of 2D Integer points

But What is a Polyhedron?

Example



2D Integer points

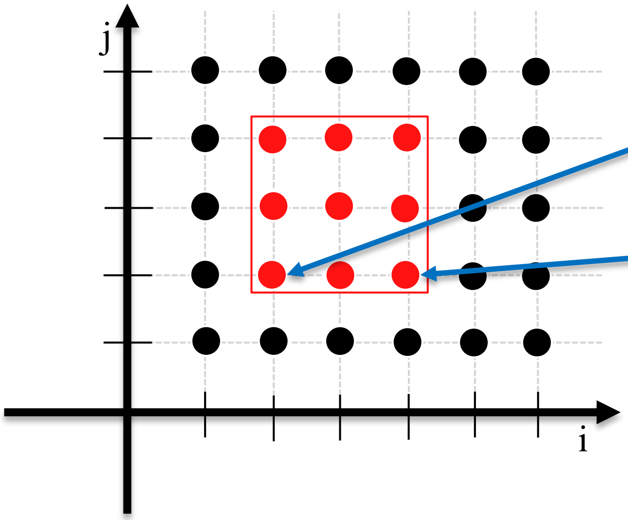
List of points

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

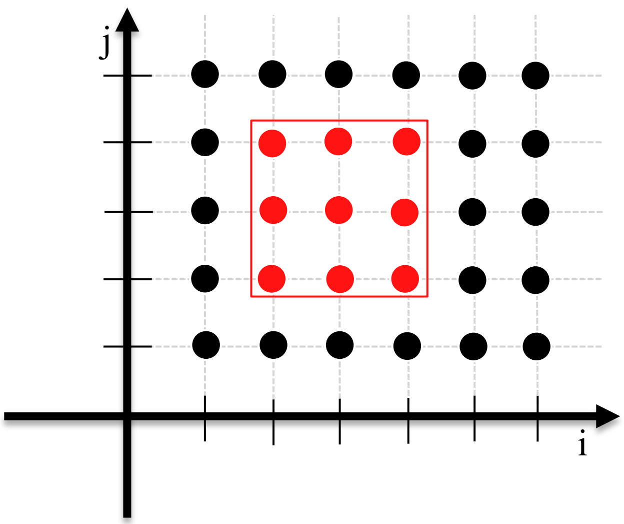
List of points

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

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

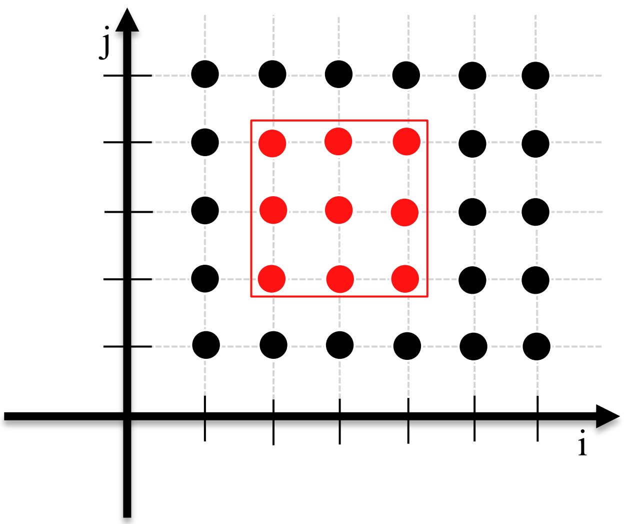
Polyhedron: described as the intersection of half-planes (e.g., $i \leq 2$), all points in the intersection are in the polyhedron

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

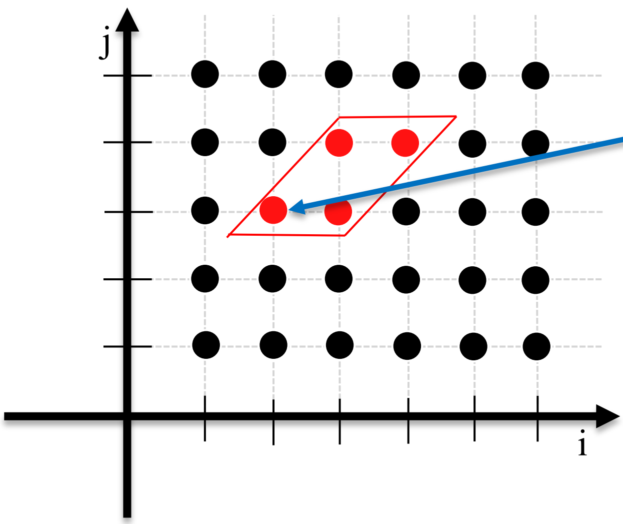
Dimensionality: 2

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

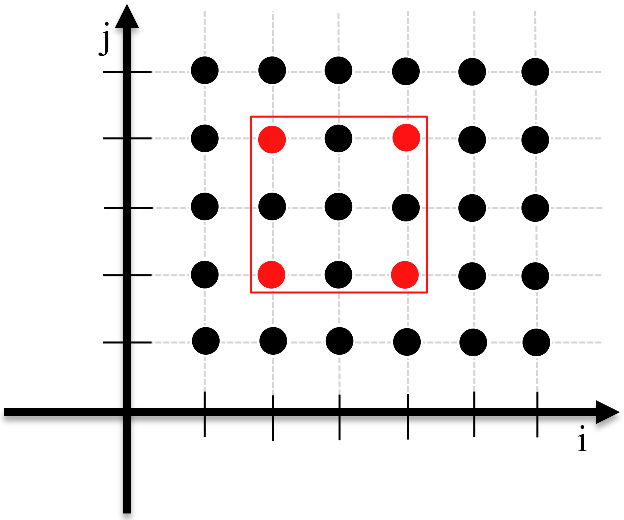
$$D : \{ [i,j] : 2 \leq i \leq 4 \text{ and } 3 \leq j \leq 4 \text{ and } j \geq i \text{ and } j \leq i+1 \}$$

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

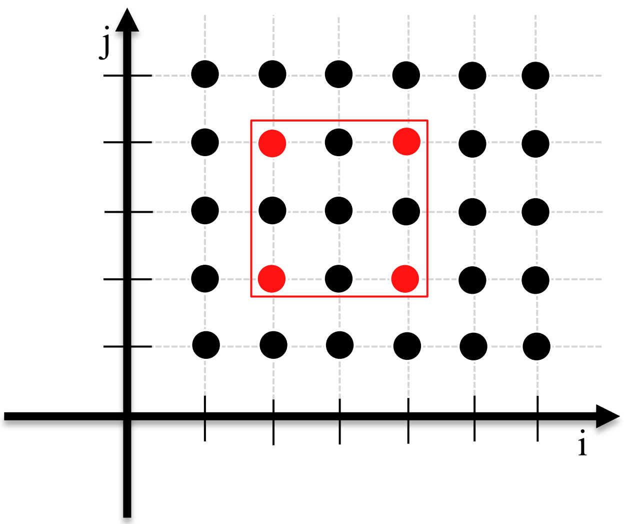
~~D : { [i,j] : 2 ≤ i ≤ 4 and 2 ≤ j ≤ 4 }~~

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 ≤ i ≤ 2 and
1 ≤ j ≤ 2 }

+

Intersected with an integer lattice:

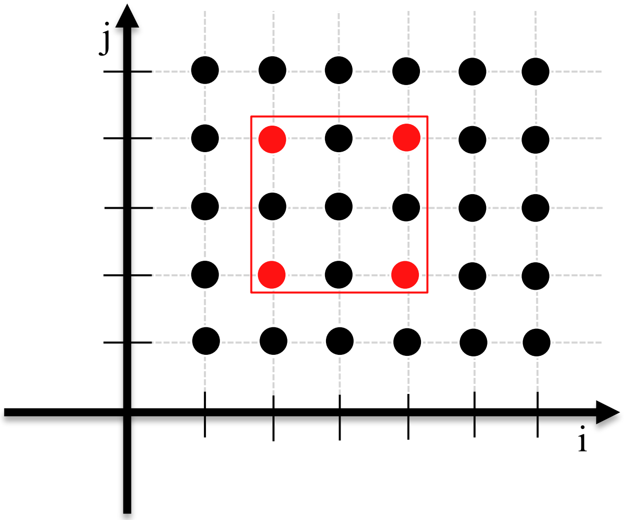
L : { [i,j] —> [x,y] : x = 2i and y = 2j }

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 $1 \leq j \leq 2$ }

\+

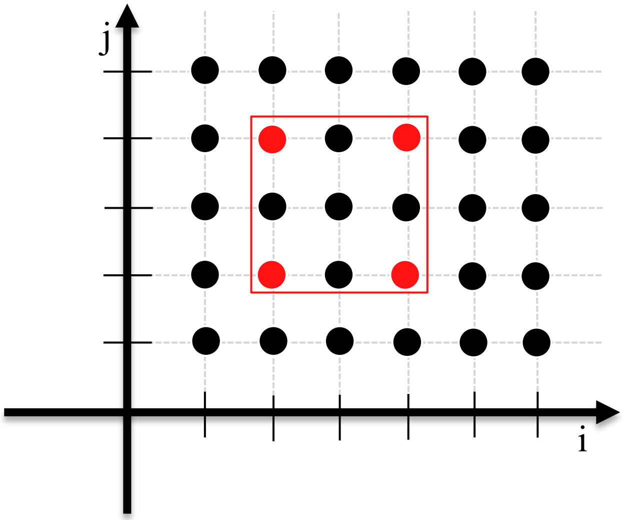
Intersected with an integer lattice:
L : { [i,j] —> [x,y] : x = 2i and y = 2j }

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:

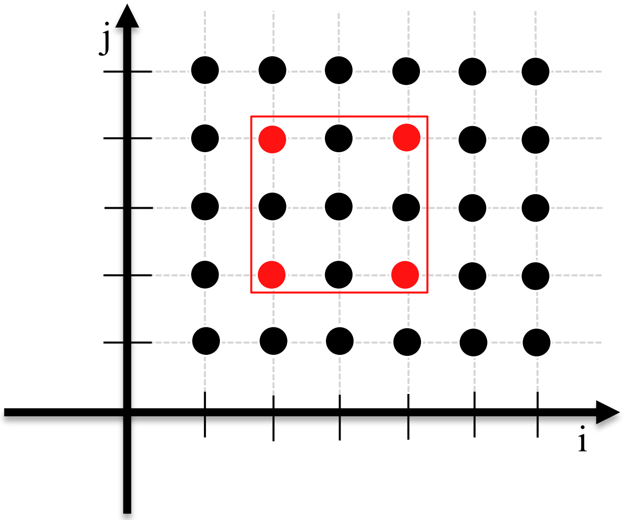
L : { [i,j] –> [x,y] : x = 2i and y = 2j }

```
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 ≤ i ≤ 2 and
1 ≤ j ≤ 2 }

+

Intersected with an integer lattice:

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

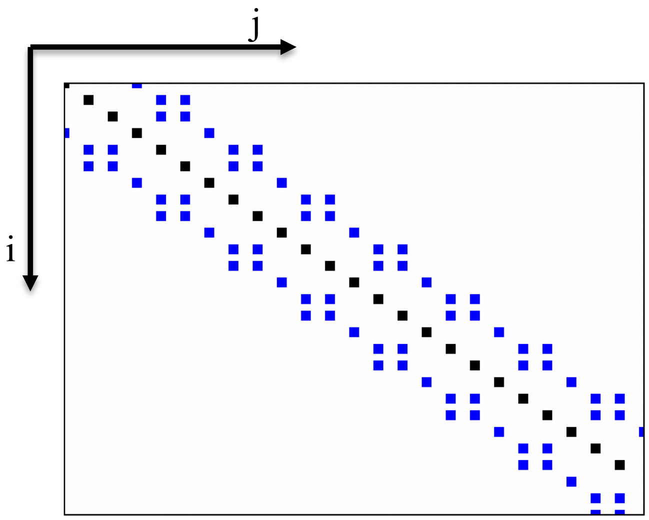
```
for (i = 1; i <= 2; i++)
  for (j = 1; j <= 2; j++)
    S(2i,2j); // x = 2i, y = 2j
```

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

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
…	…	…	…

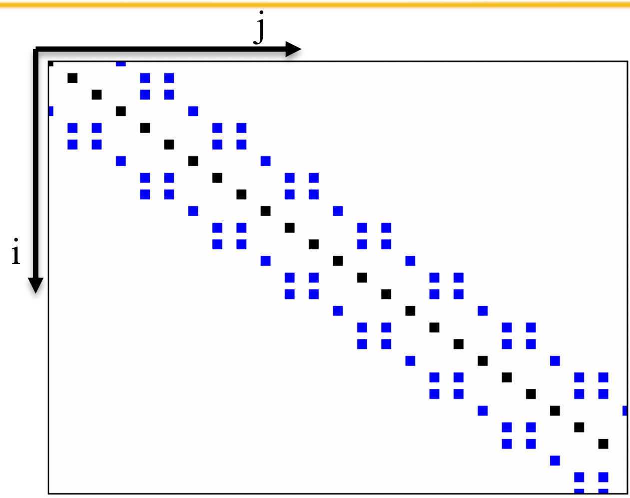
We handle sparse structures of arbitrary dimensionality, this includes sparse tensors

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?**

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 }

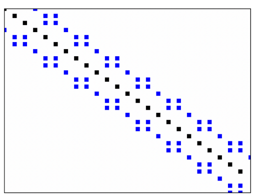
D2: { [i,j,k] : 2 <= i <= 3 and i = j and k = 16i – 12 }

When modeling problems like SpMV, we consider the trace reorderable

<u>That is, non-consecutive points in the original trace may be grouped together</u>

Complexity Trade-Offs [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**

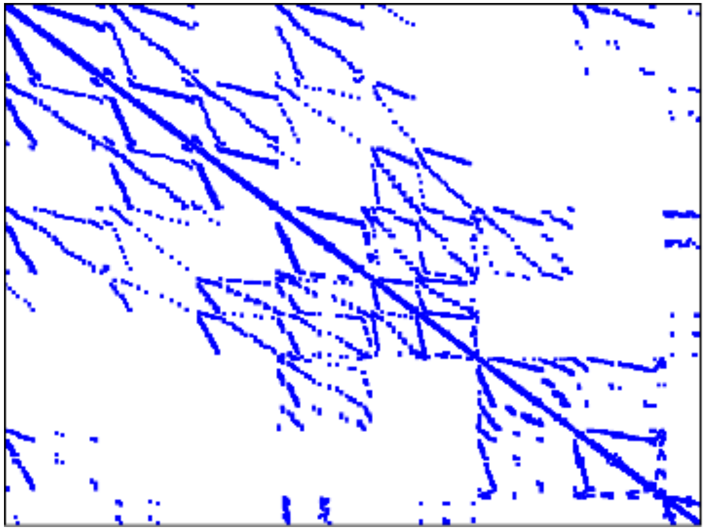


max_d	2	3	4	5	6	7	8
pieces	312	159	81	4	3	2	1
cycles	11373	11583	9938	35730	34116	39306	50371
LoC	772	1004	671	195	368	165	101

- **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

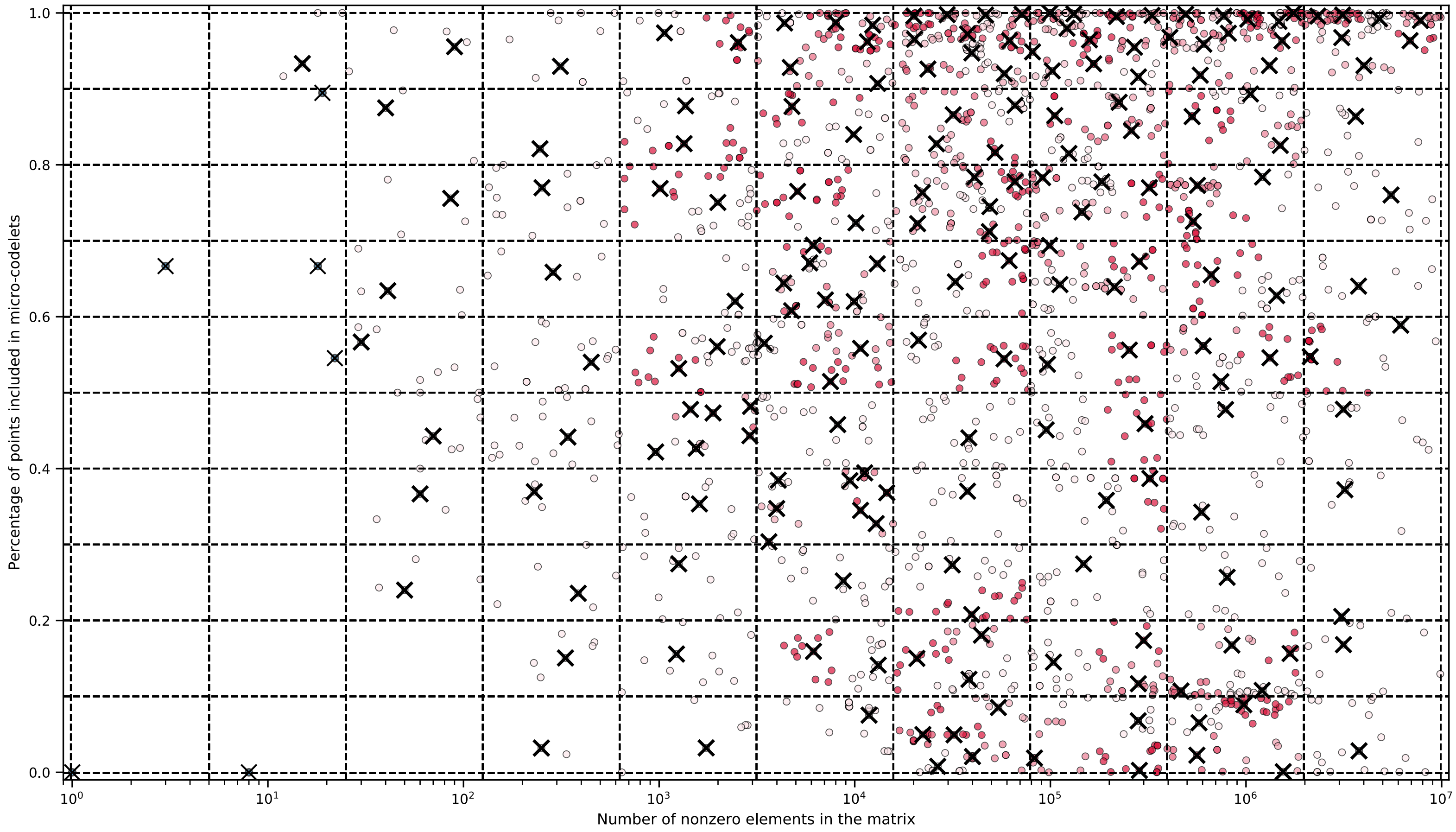


- **In this work, we design a series of algorithms that trade-off the number of polyhedra needed versus their "complexity"**
  - Try simple shape first: "rectangles", with regular strides (SIMD-friendly)
  - Try more complex shapes afterwards (skewed ones, with many dimensions)

High-Level Procedure

- 1: obtain a series of integer tuples describing the sparse structure coordinates
  - Simply scan the structure, printing the coordinates
- 2: Find simple, “rectangular” shapes by mining the trace
  - Single-level codelets: prototype shapes are chosen to be SIMD friendly
  - Implementation: mostly brute-force, but in practice extremely quick (seconds)
- 3: Try to build shapes-of-shapes, by hierarchical reconstruction
  - Create a new set of points with the polyhedra origins from 2:, and repeat!
  - Increase the complexity of shapes: use the Extended TRE algorithm for the second-level of reconstruction, as SIMD considerations are less useful here
- 4: Generate efficient code by carefully inserting code prefetch instructions
  - Code size vastly increases and exceed L1 cache, and loops often iterate over only few iterations
  - Need to explicitly prefetch the code to be executed in advance to gain performance
  - Codegen from polyhedra description is straightforwad for codelets

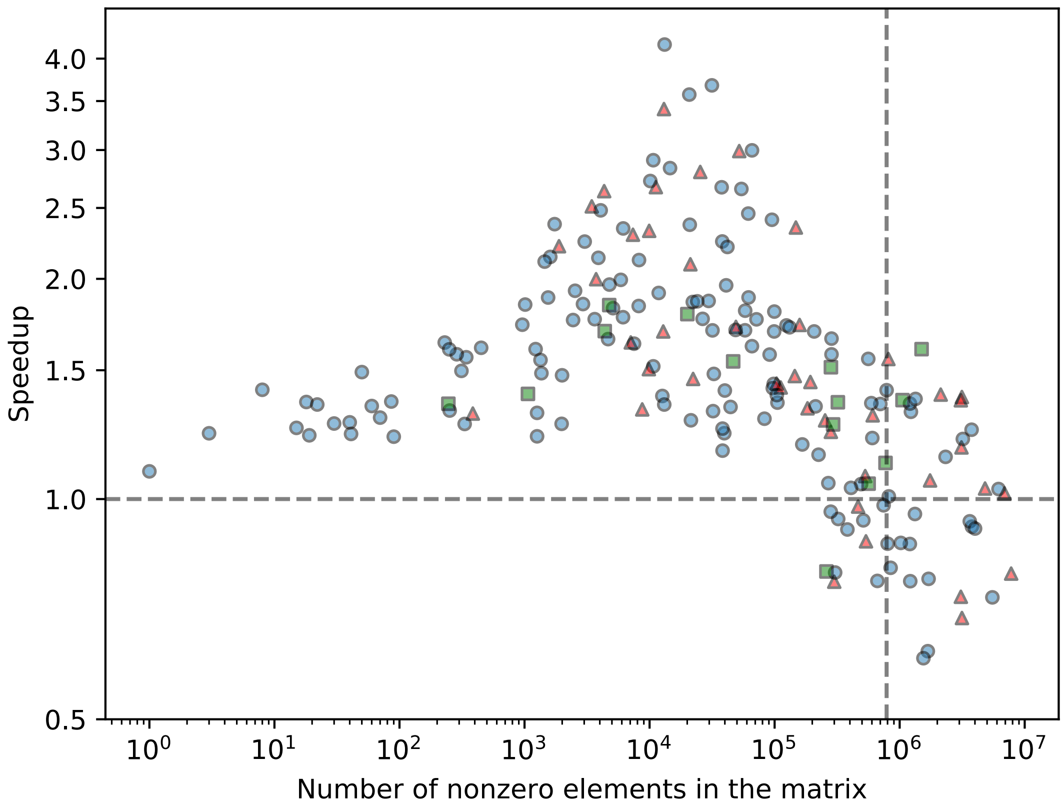
Experimental Results [1/4]



2600+ matrices from SuiteSparse with less than 10M nonzeros

We evaluate on 200 representative matrices

Experimental Results [2/4]



Experimental setup:

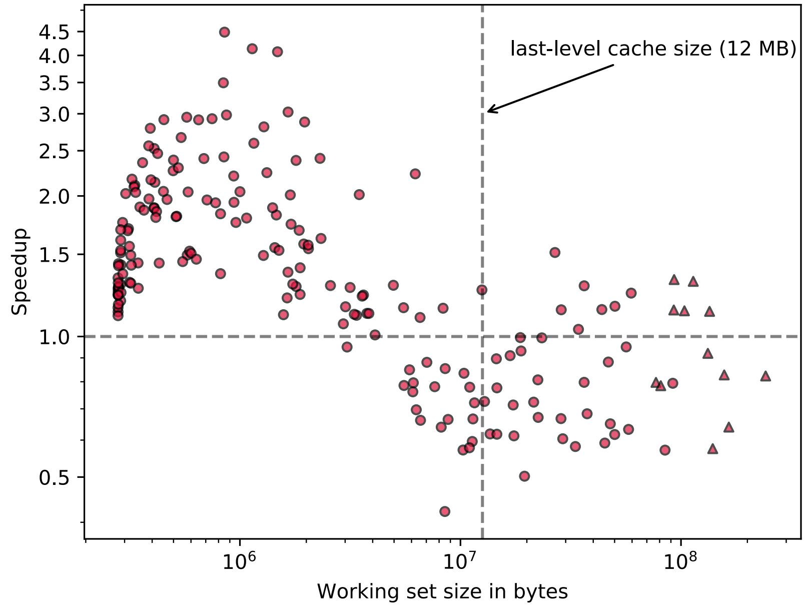
Core i7 8700k (3.7GHz)
Using hugepages
Compiled with ICC 18.03

Baselines: best of
- Vanilla SpMV C code
- Intel MKL IE

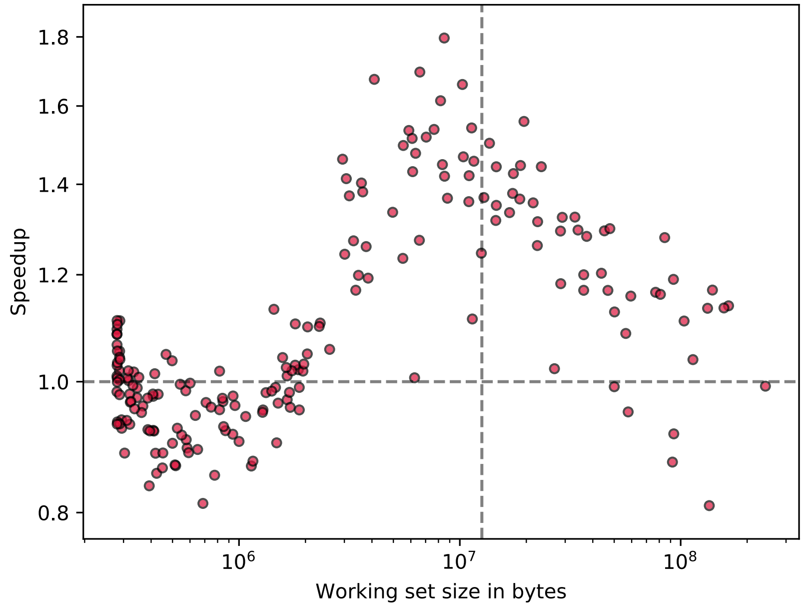
circle: single-level reconst.
triangle, square: hierarchical

- **Performance increases in the majority of cases, but not all**
  - **Complex interplay between instruction count increase, memory traffic pattern modifications, and SIMD vectorization**

Experimental Results [3/4]



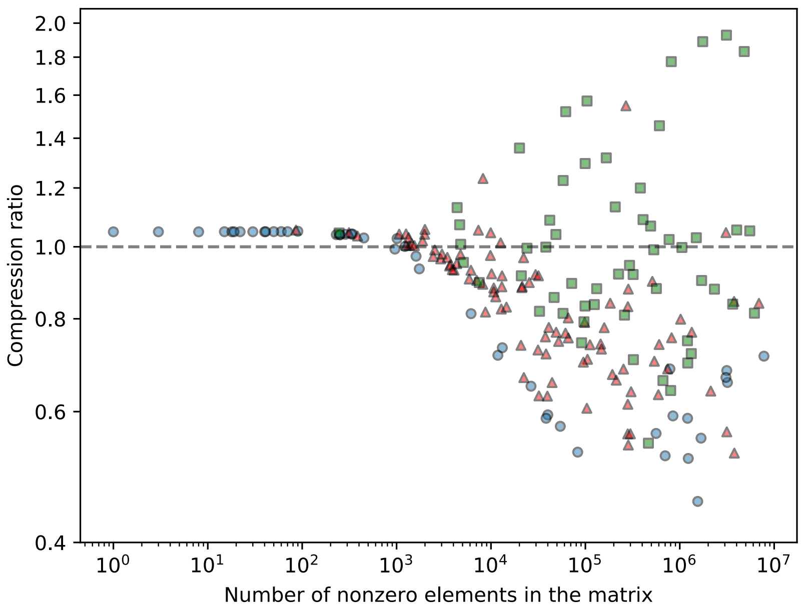
Performance **without** instruction prefetch insertion



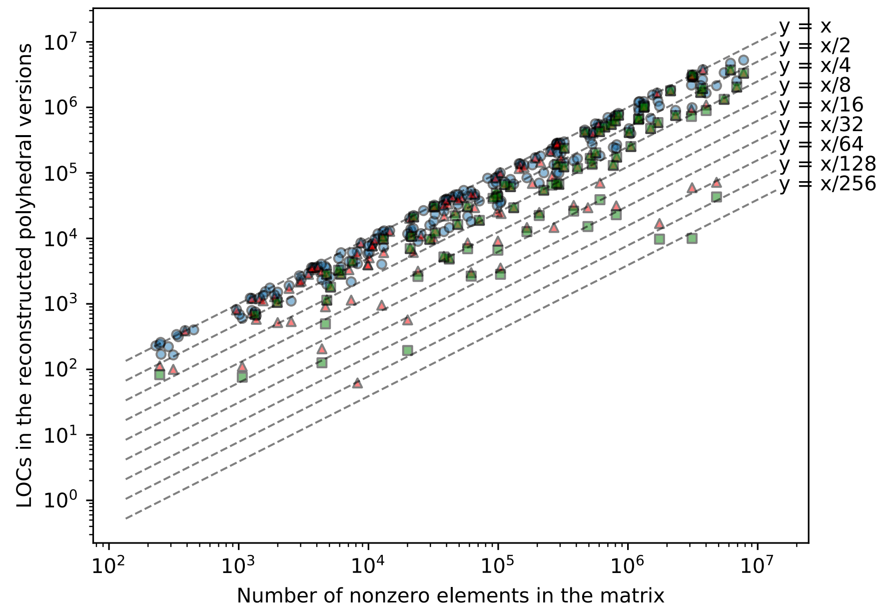
Improvement from instruction prefetch insertion alone

- **Code prefetching is critical for performance esp. for large matrices**
  - **Prefetch inserted every 64B of instructions, inserted 4kB before code is used**

Experimental Results [4/4]



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**

Take-Home Message

Sparse data structures using integer coordinates can be represented as a union of Z-polyhedra

- Performance improved, removal of indirection arrays, better SIMD
- May achieve compaction over other sparse formats, e.g. CSR

- **Quick synthesis time, but generated code can be very large**
- **General approach: works for sparse tensors**
  - Extensive study of 200 sparse matrices from SuiteSparse
  - Early results with neural network weight pruning (see paper)
- **Active line of work:**
  - Design of NN weight pruning aware of polyhedra shape objectives
  - Design new shape/polyhedron templates for better performance and compaction