Polyhedral Modeling of Immutable Sparse Matrices

Gabriel Rodríguez, Louis-Noël Pouchet

International Workshop on Polyhedral Compilation Techniques
Manchester, January 2018

Motivation and Overview

Objective: study the regularity of sparse matrices

- Can we *compress* a sparse matrix into a union of polyhedra?
- Are there *n*-dimensional polyhedra which can capture non-zeros coordinates?

Approach: affine trace compression on SpMV

- In SpMV, the i, j coordinates of non-zeros are explicit in the trace
- Reconstruct 3 streams: i, j and F_A the memory address of the data
- Trade-off between the number of polyhedra and their dimensionality

Benefits and limitations

- ▷ Enable off-the-shelf polyhedral compilation
- ▷ Performance improvements on CPU for some matrices
- ▷ *The reconstructed program requires the matrix is sparse-immutable*

Overview

Can we rewrite this . . . (CSR SpMV)

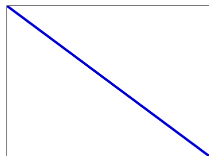
```
for( i = 0; i < N; ++i ) {
  for( j = row_start[i]; j < row_start[i+1]; ++j ) {
   y[ i ] += A_data[ j ] * x[ cols[j] ];
  }
}
```

. . . into this? (Affine SpMV): $(\mathcal{D}, F_y, F_A, F_x)$

```
for( i1 = max(..); i1 < min(..); ++i1 ) {
  ⋮
  for( in = max(..); in < min(..); ++in ) {
    y[ fy(..) ] += A_data[ fa(..) ] * x[ fx(..) ];
  }
}
```

Overview

Simple example: diagonal matrix



Variables in the sparse code

i	0	1	2	3	4	5	6	7	8	9	
cols[j]	0	1	2	3	4	5	6	7	8	9	...
j	0	1	2	3	4	5	6	7	8	9	

Nonzeros: $\mathcal{D} = \{[i,j] : 0 \leq i < N \wedge i = j\}$

Executed statements

```
y[0] = A_data[0] * x[0];
y[1] = A_data[1] * x[1];
⋮
y[N-1] = A_data[N_1] * x[N-1];
```

Overview

Simple example: diagonal matrix

Executed statements

```
y[0] = A_data[0] * x[0];
y[1] = A_data[1] * x[1];
```

```
⋮
y[N-1] = A_data[N_1] * x[N-1];
```

Affine equivalent SpMV

- Iteration domain: $\mathcal{D} = \{[i] : 0 \leq i < N\}$
- Access functions: $F_y = F_A = F_x = i$

```
for( i = 0; i < N; ++i )
  y[ i ] += A_data[ i ] * x[ i ];
```

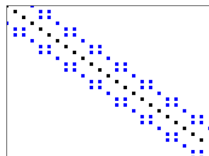
Affine equivalent SpMV

```
for( i = 0; i < N; ++i )
  y[ i ] += A_data[ i ] * x[ i ];
```

- The sparsity structure must be **immutable** across the computation.
- Note: **not necessary to copy-in data from the CSR format.**

Overview

But what about more complex examples?



Nonzero coordinates

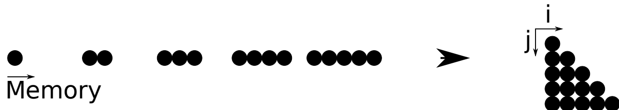
i	0	0	1	1	1	2	2	2	3	3	
cols[j]	0	3	1	4	5	2	4	5	0	3	...
j	0	1	2	3	4	5	6	7	8	9	

Affine SpMV?

```
for( i1 = max( ... ); i1 < min( ... ); ++i1 ) {
  ⋮
  for( in = max( ... ); in < min( ... ); ++in ) {
    y[ fi(i1,...,in) ] += A[ fa(...) ] * x[ fj(...) ];
  }
}
```

Code synthesis

Trace Reconstruction Engine (TRE)[1]



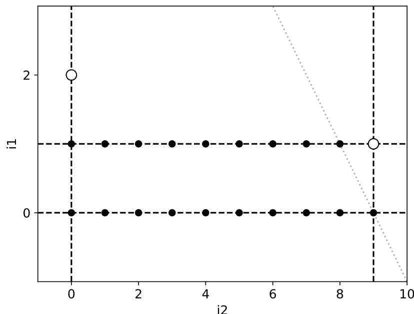
- Tool for automatic analysis of isolated memory streams.
- Generates a single, perfectly nested statement in affine loop.
  - Iteration domain $\mathcal{D}$.
  - Access function F.

[1] G. Rodríguez et al. Trace-based affine reconstruction of codes. CGO 2016.

Code synthesis

Trace Reconstruction Engine (TRE)

- Starts with simple, 2-point iteration polyhedron (1D loop).
- For each address a^k in the trace:
  - Generate lexicographical successors.
  - Accept successors accessing a^k.
  - Maybe compute new bounds for iteration polyhedron.



Code synthesis

Code generation

```
for( i = 0; i < N; ++i ) {
  for( j = pos[i]; j < pos[i+1]; ++j ) {
   y[ i ] += A_data[ j ] * x[ cols[j] ];
  }
}
```

- We inspect the input sparse matrix and generate the sequence of values of `i`, `j`, and `cols[j]` for an execution of the SpMV kernel.
- The TRE generates: $(\mathcal{D}, \mathcal{F}_y, \mathcal{F}_A, \mathcal{F}_x)$
- A simple timeout mechanism is employed to divide the trace into statements.
- TRE generates a set of statements in `scoplib` format.
- Provided to PoCC. Code generation via CLooG. No polyhedral optimization.

Output for HB/nos2

```
for (c1 = 0; c1 <= 1; c1++) {
  int __lb0 = ((-1 * c1) + 1);
  for (c3 = 0; c3 <= __lb0; c3++) {
   int __lb1 = (317 * c3);
   int __lb2 = ceild(((-2 * c1) + (-1 * c3)), 6);
   for (c5 = 0; c5 <= __lb1; c5++) {
     int __lb3 = min(floord((((-9 * c1) + (-3 * c3)) + 28), 16), ((-1 * c5) + 317));
     for (c7 = __lb2; c7 <= __lb3; c7++) {
      int __lb4 = ceild(((((4 * c1) + (5 * c3)) + (4 * c7)) + -8), 10);
      int __lb5 = min(min(floord(((((-16 * c1) + (-1 * c3)) + (-6 * c7)) + 22), 5),
                  (c1 + (2 * c3))), ((c1 + c3) + c7));
      for (c9 = __lb4; c9 <= __lb5; c9++) {
        int __lb6 = max((-1 * c7), (-1 * c9));
        int __lb7 = min(floord((((((-7 * c1) + (-1 * c3)) + (-3 * c7)) + (-2 * c9))
                    + 10), 3), ((c1 + c3) + (-1 * c9)));
        int __lb8 = max(0, (((2 * c1) + c9) + -2));
        for (c11 = __lb6; c11 <= __lb7; c11++) {
         int __lb9 = min(min(((-1 * c5) + 318), ((((-1 * c1) + (-1 * c3)) + (2 * c9))
                     + 1)), ((((-1 * c1) + (-1 * c7)) + c11) + 2));
         for (c13 = __lb8; c13 <= __lb9; c13++) {
           int __lb10 = max(max((-1 * c9), ((((c3 + (3 * c7)) + (2 * c9)) + c11) + -3)),
             (((((((3 * c1) + c3) + (3 * c7)) + (2 * c9)) + c11) + (-3 * c13)) + -3));
           int __lb11 = min(min(((c1 + (6 * c7)) + c11), ((((-4 * c1) + (-2 * c11)) +
             (-3 * c13)) + 7)), (((((3 * c1) + (-1 * c7)) + (3 * c9)) + c13) + 1));
           for (c15 = __lb10; c15 <= __lb11; c15++)
            y[+955*c1+2*c3+3*c5+1*c7+1*c9+0]=
              A[+4131*c1+5*c3+13*c5+2*c7+3*c9+1*c11+1*c13+1*c15+0]
              *x[+952*c1+2*c3+3*c5+1*c7+-2*c9+2*c11+3*c13+1*c15+0]
              +y[+955*c1+2*c3+3*c5+1*c7+1*c9+0];
}}}}}}}
```

Experimental results

Description

- Harwell-Boeing sparse matrix repository.
- Matrices which require more than $1,000$ statements are discarded during the reconstruction process.
- 242 out of 292 remain.
- 173 are ultimately converted into C code.

Reconstruction statistics

category	dims	nnz	stmts	iters	count
(0, 5]	2.47	699.56	1.43	489.42	32
(5, 20]	6.39	631.72	11.42	55.29	22
(20, 100]	6.32	1524.51	49.55	30.77	67
(100, 200]	6.29	3560.80	137.73	25.85	48
(200, 400]	6.31	7202.05	293.90	24.51	45
(400, 600]	6.40	8865.98	477.95	18.55	20
(600, 800]	6.16	17984.74	687.62	26.16	10

Experimental results

Number of statements

120
100
80
60
40
20
0

Frequency

0 100 200 300 400 500 600 700 800 900

Affine statements

Experimental results

Performance vs. Executed Instructions

bcsstm09
bcsstm25
685_bus
jagmesh1
nos1

speedup

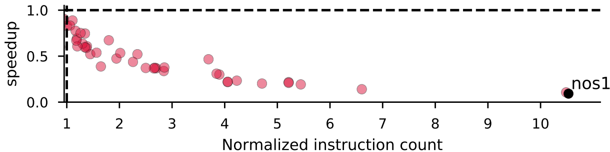
0 1 2 3 4 5 6

Normalized instruction count

0 1 2 3 4 5 6 7 8 9 10

Experimental results

More instructions, less performance

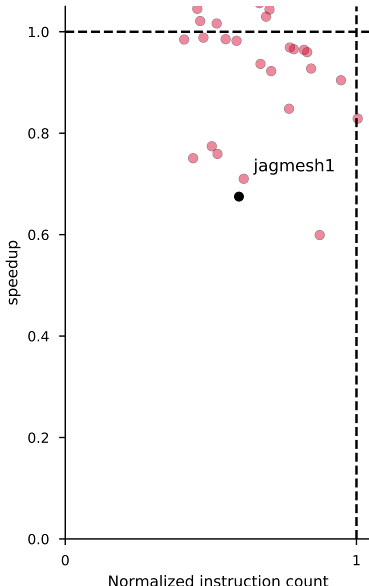


Normalized to irregular code

matrix	cycles	#insts	D1h	D1m	L2m	I1m	#branches
nos1	10.84	10.53	9.1	3.8	1.56	2.24	6.87

Experimental results

Less instructions, less performance

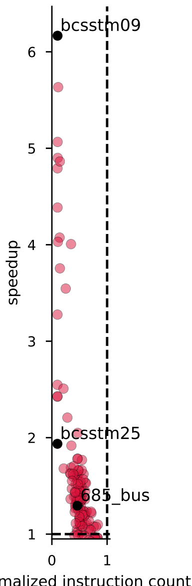


Normalized to irregular code

matrix	jagmesh1
cycles	1.48
#insts	0.60
D1h	0.77
D1m	28.95
L2m	37.88
I1m	37169.79
#branches	0.07

Experimental results

Less instructions, more performance

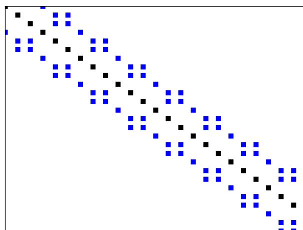


Normalized to irregular code

matrix	bcsstm09	bcsstm25	685_bus
cycles	0.16	0.52	0.77
#insts	0.10	0.10	0.46
D1h	0.17	0.01	0.99
D1m	0.00	14.44	1.09
L2m	1.31	64.75	74.55
I1m	1.09	1.48	3937.17
#branches	0.09	0.08	0.01
avx	1.00	1.00	0.00

Trade offs

Dimensionality vs. Statements vs. Performance



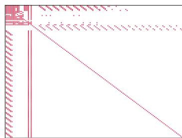
HB/nos2

$\mathbf{max_d}$	2	3	4	5	6	7	8
pieces	1273	639	321	4	3	2	1
time (s)	5.94	32	142	31	29	22	12
speedup	.98	.78	.84	.11	.11	.20	.10

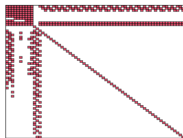
Trade offs

Density vs. Statements

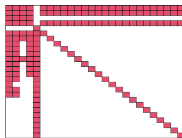
Following the sparsity structure exactly is **not** required. E.g., BCSR



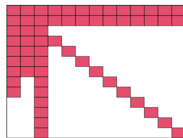
Original
31 stmts



2×2 blocks
19 stmts
$2\times$ entries



5×5 blocks
3 stmts
$3.8\times$ entries



10×10 blocks
3 stmts
$5.7\times$ entries

Future Work and Applications

Regularity exists in HB suite (292 matrices)

- Trade-off number of pieces vs. dimensionality
- TRE and trace order can be modified to generate more compact code
- Including some zero-entries can reduce code size

One possible application: sparse neural networks

- Main idea: control sparsity/connectivity to facilitate TRE's job
- Enables inference mapping to FPGA with polyhedral tools

But still requires the matrix to be sparse-immutable

- In essence, this is *data-specific compilation*
- Neural nets, road networks, etc. qualify

Take-Home Message

Regularity in sparse matrices can be automatically discovered

- Trace reconstruction on SpMV gives polyhedral-only representation of the matrix
- But the number and size of pieces may render the process useless

Affine SpMV code can be automatically generated

- Simple scanning of the rebuilt polyhedra
- This work: only looking at single-core CPUs, no transformation
- But enables off-the-shelf polyhedral compilation

Possible applications require sparse-immutable matrices

- Not an issue for many situations (e.g., inference of neural nets)
- The benefits depend on the sparsity pattern
    - Best situation: control both sparsity creation and TRE simultaneously

Polyhedral Modeling of Immutable Sparse Matrices

Gabriel Rodríguez, Louis-Noël Pouchet

International Workshop on Polyhedral Compilation Techniques
Manchester, January 2018