Polyhedral Modeling of Immutable Sparse Matrices

Gabriel Rodríguez, Louis-Noël Pouchet

UNIVERSIDADE DA CORUÑA  
Colorado State University

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)

```c
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): \((D, F_y, F_A, F_x)\)

```c
for ( il = max(..); il < min(..); ++il ) {
    ...
    for ( in = max(..); in < min(..); ++in ) {
        y[ fy(..) ] += A_data[ fa(..) ] * x[ fx(..) ];
    }
}
```
Overview
Simple example: diagonal matrix

![Diagonal Matrix Diagram]

<table>
<thead>
<tr>
<th>Variables in the sparse code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
</tr>
<tr>
<td>cols[j]</td>
</tr>
<tr>
<td>$j$</td>
</tr>
</tbody>
</table>

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

Executed statements

```
| y[0] = A_data[0] * x[0];
| ...;
| y[N-1] = A_data[N-1] * x[N-1];
```
Overview

Simple example: diagonal matrix

Executed statements

\[
\begin{align*}
y[0] & = A_{\text{data}}[0] \times x[0]; \\
y[1] & = A_{\text{data}}[1] \times x[1]; \\
& \vdots \\
y[N-1] & = A_{\text{data}}[N-1] \times x[N-1];
\end{align*}
\]

Affine equivalent SpMV

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

\[
\text{for ( } i = 0; i < N; ++i \text{ )} \\
y[ i ] += A_{\text{data}}[ i ] \times x[ i ];
\]
Affine equivalent SpMV

```c
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

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cols[j]</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Affine SpMV?

```
for ( il = max( ... ); il < min( ... ); ++il ) {
  :
  for ( in = max( ... ); in < min( ... ); ++in ) {
    y[ fi(il,...,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.
We inspect the input sparse matrix and generate the sequence of values of $i$, $j$, and $\text{cols}[j]$ for an execution of the SpMV kernel.

The TRE generates: $(D, F_y, F_A, F_x)$

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 = ceil(((-2 * c1) + (-1 * c3)), 6);
        for (c5 = 0; c5 <= __lb1; c5++) {
            int __lb3 = min(floor((-9 * c1) + (-3 * c3) + 28), 16), ((-1 * c5) + 317));
            for (c7 = __lb2; c7 <= __lb3; c7++) {
                int __lb4 = ceil(((4 * c1) + (5 * c3)) + (4 * c7)) + 8), 10);
                int __lb5 = min(min(floor(((-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(floor(((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(floor((-9 * c1) + (-3 * c3) + 28), 16), ((-1 * c5) + 317));
                            for (c13 = __lb8; c13 <= __lb9; c13++) {
                                int __lb10 = max(max((-1 * c9), ((c3 + (3 * c7)) + (2 * c9)) + c11) + 3), ((c1 + c3) + (2 * c9)) + 3), ((c1 + c3) + (3 * c7)) + 3), ((c1 + c3) + (6 * c7)) + 3), ((c1 + (6 * c7)) + c11), ((-4 * c1) + (-2 * c11)) + (3 * c13)) + 7), ((c1 + (6 * c7)) + c11), ((c1 + c3) + (3 * 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

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

Number of statements
Experimental results

Performance vs. Executed Instructions

- nos1
- jagmesh1
- bcsstm09
- bcsstm25
- 685_bus
Experimental results
More instructions, less performance

![Graph showing speedup versus normalized instruction count]

<table>
<thead>
<tr>
<th>matrix</th>
<th>cycles</th>
<th>#insts</th>
<th>D1h</th>
<th>D1m</th>
<th>L2m</th>
<th>l1m</th>
<th>#branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>nos1</td>
<td>10.84</td>
<td>10.53</td>
<td>9.1</td>
<td>3.8</td>
<td>1.56</td>
<td>2.24</td>
<td>6.87</td>
</tr>
</tbody>
</table>
Experimental results

Less instructions, less performance

Normalized instruction count

<table>
<thead>
<tr>
<th>matrix</th>
<th>jagmesh1</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>1.48</td>
</tr>
<tr>
<td>#insts</td>
<td>0.60</td>
</tr>
<tr>
<td>D1h</td>
<td>0.77</td>
</tr>
<tr>
<td>D1m</td>
<td>28.95</td>
</tr>
<tr>
<td>L2m</td>
<td>37.88</td>
</tr>
<tr>
<td>I1m</td>
<td>37169.79</td>
</tr>
<tr>
<td>#branches</td>
<td>0.07</td>
</tr>
</tbody>
</table>
**Experimental results**

Less instructions, more performance

---

**Normalized to irregular code**

<table>
<thead>
<tr>
<th>matrix</th>
<th>bcsstm09</th>
<th>bcsstm25</th>
<th>685_bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>0.16</td>
<td>0.52</td>
<td>0.77</td>
</tr>
<tr>
<td>#insts</td>
<td>0.10</td>
<td>0.10</td>
<td>0.46</td>
</tr>
<tr>
<td>D1h</td>
<td>0.17</td>
<td>0.01</td>
<td>0.99</td>
</tr>
<tr>
<td>D1m</td>
<td>0.00</td>
<td>14.44</td>
<td>1.09</td>
</tr>
<tr>
<td>L2m</td>
<td>1.31</td>
<td>64.75</td>
<td>74.55</td>
</tr>
<tr>
<td>L1m</td>
<td>1.09</td>
<td>1.48</td>
<td>3937.17</td>
</tr>
<tr>
<td>#branches</td>
<td>0.09</td>
<td>0.08</td>
<td>0.01</td>
</tr>
<tr>
<td>avx</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

---

Normalized instruction count

speedup

bcsstm09

bcsstm25

685_bus
Trade offs
Dimensionality vs. Statements vs. Performance

<table>
<thead>
<tr>
<th>$\text{max}_d$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>pieces</td>
<td>1273</td>
<td>639</td>
<td>321</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>time (s)</td>
<td>5.94</td>
<td>32</td>
<td>142</td>
<td>31</td>
<td>29</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>speedup</td>
<td>.98</td>
<td>.78</td>
<td>.84</td>
<td>.11</td>
<td>.11</td>
<td>.20</td>
<td>.10</td>
</tr>
</tbody>
</table>
Trade offs
Density vs. Statements

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

Original
31 stmts

2 \times 2 \text{ blocks}
19 stmts
2\times \text{ entries}

5 \times 5 \text{ blocks}
3 stmts
3.8\times \text{ entries}

10 \times 10 \text{ blocks}
3 stmts
5.7\times \text{ 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