Abstractions for Specifying Sparse Matrix Data Transformations

Payal Nandy
Mary Hall
University of Utah

Eddie C. Davis
Catherine Olschanowsky
Boise State University

Mahdi S Mohammadi, Wei He
Michelle Strout
University of Arizona
Motivation

- The polyhedral model is suitable for **affine**
  - loop bounds, array access expressions and transformations
- Polyhedral model unsuitable for sparse matrix & unstructured mesh computations (**non-affine**)
  - Array accesses of the form $A[B[i]]$
  - Loop bounds of the form $\text{index}[i] \leq j < \text{index}[i+1]$

- Key Observation
  - *Compiler generated code for run time inspector & executor*
  - *Run time inspection*
    - can reveal mapping of iterations to array indices
    - Potentially change iteration or data space
## Related Work

### Inspector/Executor
- Mirchandaney, Saltz et al., 1988
- Rauchwerger, 1998
- Basumallik and Eigenmann, 2006
- Ravishankar et al., 2012

### Polyhedral Support for Indirection
- Pugh and Wonnacott, 1994

### Frameworks for Sparse Computations
- SIPR: Shpeisman, 1999
- Bernoulli: Mateev, 2001

### Data Transformations
- Bik, 1996
- Ding and Kennedy, 1999
- Mellor-Crummey et al., 2001
- Gilad et al., 2010
- van derSpek, 2011

Prior work did not integrate all of these, and mostly did not expand data with zero-valued elements.
CHiLL-I/E - Vision
Foundation – Sparse Polyhedral Framework

• Loop transformation framework built on the polyhedral model
• Uses uninterpreted functions to represent index arrays
• Enables the composition of inspector-executor transformations
• Exposes opportunities for compiler to
  – Simplify indirect array accesses and
  – Optimize inspector-executor code
Foundation – CHiLL Compiler Framework

• Runtime data & iteration reordering transformations for non-affine loop bounds and array access
  – Make-dense
  – Compact, compact-and-pad
• Composable with polyhedral transformations
  – Tile, skew, permute
• Integration with user-specified Inspectors
• Automatically generated Inspector/Executors
  – Inspectors optimized for making less passes over data
  – Optimized executors performed comparable to runtime libraries

[CGO ‘14], [PLDI ‘15] [SC ‘16] [IPDPS ‘16] [LCPC ‘16]
Prior Research Performance Indicators

Performance of Compiler generated Inspectors and Executors competitive with CUSP

DIA Inspector Speedup

DIA Executor Performance

[PLDI’15]
Contribution

• Derive abstractions for Sparse Matrix *Data Transformations*  
  – Focus on transformations that modify data representation

• Extend Sparse Polyhedral Framework to Support data transformations  
  – Modify data representation to reflect structure of input matrix  
  – Expand iteration space to match new data representation

• Generalize representation of Inspector/executor transformations  
  – Goal: automatically compose them
Abstractions

Transformation Relations
- Include uninterpreted functions
- Includes non-affine transformations
- Composable with existing transformations

Inspector Dependence Graph
- Derived from Transformation relations
- Data flow representation of Inspector functionality

Automatic Generation of optimized Inspector/Executor
- Compiler walks IDG to generate Inspector
- Inspector instantiates explicit functions for Executor
Sparse Matrix-Vector Multiply (SpMV)

Begin with Compressed Sparse Row (CSR) format

\[ A: \begin{bmatrix} 1 & 5 & 7 & 2 & 3 & 6 & 4 \end{bmatrix} \]
\[ \text{index:} \begin{bmatrix} 0 & 2 & 4 & 6 & 7 \end{bmatrix} \]
\[ \text{col:} \begin{bmatrix} 0 & 1 & 0 & 1 & 2 & 3 & 3 \end{bmatrix} \]

Compressed Sparse Row (CSR)

for (i=0; i < n; i++)
  for (j=index[i]; j<index[i+1]; j++)
    y[i]+=A[j]*x[col[j]];
Sparse Matrix Formats

**Iteration Space Transformation**

**Data & Iteration Space Transformation**

A: \[\begin{bmatrix} 1 & 5 & 7 & 2 & 3 & 6 & 4 \end{bmatrix}\]

row: \[\begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 3 \end{bmatrix}\]

col: \[\begin{bmatrix} 0 & 1 & 0 & 1 & 2 & 3 & 3 \end{bmatrix}\]

**COO**

**DIA**

**BCSR**

**ELL**

*Moldyn (molecular dynamics) – Data + Iteration Reordering*
CSR to COO

Transformation Relations
\[ T_{\text{coalesce}} = \{[i,j] \rightarrow [k] \mid k = c(i,j) \ 0 \leq k < \text{NNZ} \} \]

\[ I_{\text{exec}} = T_{\text{coalesce}}(I) \]

Generate Inspector
\[ \text{NNZ} = \text{count}(I) \]
\[ c = \text{order}(I) \]
\[ c_{\text{inv}} = \text{invert}(c) \]

Inspector

Executor
Enabling Data Transformations

**make-dense**

for \( i = 0; \ i < n; \ i++ \)
for \( j = \text{index}[i]; \ j < \text{index}[i+1]; \ j++ \)
\[ y[i] += A[j] \times x[\text{col}[j]] ; \]

for \( i = 0; \ i < n; \ i++ \)
for \( k = 0; \ k < n; \ k++ \)
for \( j = \text{index}[i]; \ j < \text{index}[i+1]; \ j++ \)
if \( k == \text{col}[j] \)
\[ y[i] += A[j] \times x[k] \]
CSR to DIA: Transformations

**Dense Matrix**

\[
\begin{bmatrix}
1 & 5 & 0 & 0 \\
7 & 2 & 0 & 0 \\
0 & 0 & 3 & 6 \\
0 & 0 & 0 & 4
\end{bmatrix}
\]

**CSR Format**

\[A \begin{bmatrix} 1 & 5 & 7 & 2 & 3 & 6 & 4 \end{bmatrix}
\]

index \[\begin{bmatrix} 0 & 2 & 4 & 6 & 7 \end{bmatrix}\]

col \[\begin{bmatrix} 0 & 1 & 0 & 1 & 2 & 3 & 3 \end{bmatrix}\]

**DIA Format**

\[A' \begin{bmatrix} 0 & 1 & 5 \\
7 & 2 & 0 \\
0 & 3 & 6 \\
0 & 4 & 0 \end{bmatrix}\]

offsets \[\begin{bmatrix} -1 \ 0 \ 1 \end{bmatrix}\]
# Compact-and-pad

<table>
<thead>
<tr>
<th></th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>X</td>
</tr>
</tbody>
</table>

|   | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
|   | X | X | X | X | X | • | X | X | • | X | X | • | X | X | X | • | X | X | • | X | X |

- **Eliminate entirely**
- **Pad with 0**
- **Eliminate entirely**

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>•</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>•</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**CSR to DIA**

**Transformation Relations**

\[
T_{\text{make-dense}} = \{[i,j] \rightarrow [i,k,j] \mid 0 \leq k < N \wedge k = \text{col}(j) \}
\]

\[
T_{\text{skew}} = \{[i,k,j] \rightarrow [i, k',j] \mid k' = k-i \}
\]

\[
T_{\text{compact-and-pad}} = \{[k'.i,j] \rightarrow [i;d] \mid 0 \leq d < ND \wedge k' = \text{col}(j) - i \wedge c(d) = k' \}
\]

\[
\text{Iexec} = T_{\text{compact-and-pad}}(T_{\text{skew}}(T_{\text{make-dense}}(I)))
\]

**Generate Inspector**

\[
D_{\text{set}} = \{[k'] \mid \exists j, k' = \text{col}(j)-i \wedge \text{index}(i) \leq j < \text{index}(i+1)\}
\]

\[
ND = \text{count}(D_{\text{set}})
\]

\[
C = \text{order}(D_{\text{set}})
\]

\[
A_{\text{prime}} = \text{calloc}(N*ND*\text{sizeof(datatype)})
\]

\[
\text{map}: R_A \rightarrow A_{\text{prime}} = \{[j] \rightarrow [i,d] \mid 0 \leq d < ND \exists k', k' = \text{col}(j)-i \wedge c(d)=k' \}
\]
Inspector Code for DIA

\[
\begin{align*}
ND &= 0; \quad D\_set = \text{emptyset}; \\
\text{for}(i = 0; \ i < N; \ i++) \\
&\quad \text{for}(j = \text{index}[i]; \ j < \text{index}[i+1]; \ j++) \\
&\quad \quad k\_prime = \text{col}(j) - i; \\
&\quad \quad \text{if (!marked}[k\_prime]) \\
&\quad \quad \quad D\_set = D\_set \cup <k\_prime,ND++>; \\
&\quad \}
A\_prime = \text{calloc}(N*ND*\text{sizeof(datatype)}); \\
c = \text{calloc}(ND*\text{sizeof(indextype)}); \\
\text{for}(i = 0; \ i < N; \ i++) \\
&\quad \text{for}(j = \text{index}[i]; \ j < \text{index}[i+1]; \ j++) \\
&\quad \quad k\_prime = \text{col}(j) - i; \\
&\quad \quad d = \text{lookup}(k\_prime, D\_set); \\
&\quad \quad c[d] = k\_prime; \\
&\quad \quad A\_prime[i][d] = A[j]; \\
&\}
\end{align*}
\]

Executor Code

\[
\begin{align*}
\text{for } (i=0; \ i < N; \ i++) \\
&\quad \text{for}(d=0; \ d < ND; \ d++) \\
&\quad \quad y[i] += A[i][d]*x[i+c[d]]; \\
\end{align*}
\]
Future Work - Optimizing the IDG

• Minimize inspector passes over input data
• Extend IDG to support fusion of Inspectors
• Additional optimizations
  – Dynamic data structures (e.g. linked lists) to eliminate sweeps to calculate size of data representation
  – Integrate existing inspector library functions
Conclusion

• Abstractions for data transformations in sparse matrix & unstructured mesh computations

• Approach
  – Transformation Relations
  – Inspector Dependence Graph
  – Compiler generated optimized Inspector/Executor code

• Vision: Create a framework to compose complex transformation sequences for inspectors and executors