Sympiler: Inspection and Transformation of Sparse Matrix Computations

Maryam Mehri Dehnavi

University of Toronto
The **sparsity patterns** in many applications such as power modeling, animation, and circuit simulation often arises from the physical topology of the underlying system, the discretization, and the governing equations. Information about the sparsity pattern is called *symbolic information*.
**EXAMPLE: CHOLESKY FACTORIZATION**

**Cholesky factorization** is commonly used in direct solvers and is used to precondition iterative solvers.

**The elimination tree (T)** is one of the most important graph structures used in the symbolic analysis of sparse factorization algorithms.
Sympiler* is a domain-specific compiler for generating high-performance code for sparse solvers.

- It uses symbolic information to transform the sparse code.
- Sympiler applies prune and block transformations but does not support parallelism for multicore.

*Cheshmi, Kamil, Strout, Dehnavi "Sympiler: transforming sparse matrix codes by decoupling symbolic analysis." SC17
**ParSy (Parallel Sympiler)**

- **Sympiler** is a domain-specific compiler for generating high-performance code for sparse solvers.
  - It uses symbolic information to transform the sparse code.
  - Sympiler applies prune and block transformations but does not support parallelism for multicore.

- **ParSy** generates parallel code for sparse matrix computations.
  - ParSy is built on top of Sympiler.
  - However, it can also be implemented at run-time.

*Cheshmi, Kamil, Strout, Dehnavi "ParSy: Inspection and Transformation of Sparse Matrix Computations for Parallelism." SC18*
PARSY (AND SYMPILER) INTERNALS

- Sparsity Pattern
- Numerical Method
- Symbolic Inspector
- Inspector-Guided Transformations
- Code Generation
PARSy (and Sympiler) Internals

Inputs

- Sparsity Pattern
- Numerical Method

Symbolic Inspector

Inspector-Guided Transformations

Code Generation
PARSY (AND SYMPILER) INTERNALS

Inspection of the dependence graph to create *inspection sets*

- Sparsity Pattern
- Numerical Method
- Inspector-Guided Transformations
- Code Generation

Symbolic Inspector
The *inspection sets* are used to transform the sparse code
Example: ParSy introduces **H-Level inspection** and **H-Level transformation** to generate parallel code.
During symbolic inspection, ParSy creates an **H-Level set** by inspecting the dependence graph using the **Load-Balanced Level Coarsening** algorithm. The result of inspection is the **H-level set**.

\[
\text{H-Level Set} = \{\{1, 2, 3, 4, 5\}, \{6, 7, 8\}, \{10, 11, 9, 12\}\} ; \{\{13, 14, 15\}\} ; }
\]

**Inspection Graph:** Elimination Tree

**Inspection Strategy:**
- H-Level inspection
- (Load-Balanced Level Coarsening)

**Inspection Set:**
- H-Level set
Hierarchical Level (H-Level) transformation transforms ParSy’s internally annotated code using the H-Level set to generate parallel code.

H-Level:
```c
for(I_1) {
    ...
    for(I_n(I_1)) {
        Atomic:
        c /= a[idx(I_1,...,I_n)];
    }
}
```

Internally annotated code

Transformed with H-Level:
```c
for ( every l–partition l ) {
    #pragma omp parallel for private(pVars)
    for ( every w–partition w ) {
        for ( every v ∈ HLevelSet[l][w] ) {
            I_1 = v
            ...
            for(I_n(I_1)) {
                #pragma omp atomic
                c /= a[idx(I_1,...,I_n)];
            }
        }
    }
}
```

Hierarchical Level (H-Level) transformation
**THE PARSy-GENERATED CODE FOR CHOLESKY**

**H-Level:**

```c
for (int i=0; i<blockNo; ++i){
    b1 = block2col[i]; b2 = block2col[i+1];
    f = A(:,b1:b2);
    // Update phase
    for(block r=0 to i-1 L(i,r)!=0){
        f = GEMM(L(b1:n,r),transpose(L(i,r)));
    }
    // Diagonal operation
    L(b1:b2,b1:b2)=POTRF(f(b1:b2));
    // Off-diagonal operations
    for(off-diagonal elements in f){
        L(b2+1:n,b1:b1) = TRSM(f(b1+1:n,b1:b2),L(b1:b2,b1:b2));
    }
}
```

**for**(every l-partition l < nlevels-1){
```
#pragma omp parallel for private(f){
    for(every w-partition w){
        for(every v ∈ HLevelSet[l][w]){
            int i = v;
            b1 = block2col[i]; b2 = block2col[i+1];
            f = A(:,b1:b2);
            for(block r=0 to i-1 L(i,r)!=0){
                f = GEMM(L(b1:n,r),transpose(L(i,r)));
            }
            L(b1:b2,b1:b2)=POTRF(f(b1:b2));
            for(off-diagonal elements in f){
                L(b2+1:n,b1:b1) = TRSM(f(b1+1:n,b1:b2),L(b1:b2,b1:b2));
            }
        }
    }
}
```

//Specialized code for the last l-partition.
Chol_Specialized(HLevelSet[nevels - 1][0]);
```
### Experimental Setup

Numeric and symbolic times are compared separately where applicable; **Target processor**: Intel® Xeon® Platinum 8160 (Skylake); **Benchmarks**: Suitesparse matrix collection

<table>
<thead>
<tr>
<th>Name</th>
<th>Application</th>
<th>Order ($10^3$)</th>
<th>Non-zeros ($10^6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3_circuit</td>
<td>Circuit simulation</td>
<td>1585</td>
<td>127.3</td>
</tr>
<tr>
<td>StocF_1465</td>
<td>Computational fluid dynamics problem</td>
<td>1465.1</td>
<td>1245</td>
</tr>
<tr>
<td>Hook_1498</td>
<td>3D mechanical problem</td>
<td>1498</td>
<td>1783.8</td>
</tr>
<tr>
<td>audikw_1</td>
<td>Structural problem</td>
<td>943.7</td>
<td>1473.1</td>
</tr>
<tr>
<td>bone010</td>
<td>Model reduction problem</td>
<td>986.8</td>
<td>1210.1</td>
</tr>
<tr>
<td>Emilia_923</td>
<td>Geomechanical model</td>
<td>923.1</td>
<td>1992</td>
</tr>
<tr>
<td>Fault_639</td>
<td>Contact mechanics</td>
<td>638.8</td>
<td>1275.4</td>
</tr>
<tr>
<td>nd24k</td>
<td>2D/3D problem</td>
<td>72</td>
<td>435.9</td>
</tr>
</tbody>
</table>
PARSy vs Libraries: Cholesky

GFLOP/s

- MKL Pardiso (Metis)
- ParSy(Metis)
- Pastix (Scotch)
- ParSy(Scotch)

G3 circuit
StocF-1465
Hook_1498
audikw_1
bone010
Emilia_923
Fault_639
nd24k

2.2X 1.0X 1.3X 1.5X 1.6X 1.2X 1.1X 1.4X 1.8X 1.5X 1.7X 1.4X 1.3X
**PARSy vs Libraries: Triangular Solve**

![Graph showing performance comparison between MKL and ParSy for various libraries.](image-url)

- **GFLOP/s**
  - G3_circuit: 1.2X
  - StocF-1465: 3.0X
  - Hook_1498: 2.4X
  - audtkw_1: 5.1X
  - bone010: 3.0X
  - Emilia_923: 2.0X
  - Fault_639: 2.6X
  - nd24k: 5.3X

- **Libraries Compared:** TBLIBRARIES
  - MKL: Triangular Solve
### Inspection Overhead: Cholesky

<table>
<thead>
<tr>
<th>Name</th>
<th>MKL Acc Time / ParSy Acc Time</th>
<th>Pastix Acc Time / ParSy Acc Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3_circuit</td>
<td>1.4</td>
<td>0.9</td>
</tr>
<tr>
<td>StocF_1465</td>
<td>1.4</td>
<td>0.9</td>
</tr>
<tr>
<td>Hook_1498</td>
<td>1.25</td>
<td>1</td>
</tr>
<tr>
<td>audikw_1</td>
<td>1.25</td>
<td>1</td>
</tr>
<tr>
<td>bone010</td>
<td>1.11</td>
<td>0.9</td>
</tr>
<tr>
<td>Emilia_923</td>
<td>1.4</td>
<td>1.25</td>
</tr>
<tr>
<td>Fault_639</td>
<td>1.25</td>
<td>1.25</td>
</tr>
<tr>
<td>nd24k</td>
<td>1.25</td>
<td>1</td>
</tr>
</tbody>
</table>

Acc Time = Symbolic analysis time + Numerical factorization
Support other kernels with the long-term objective of optimizing across-kernels.

- Extend ParSy/Sympiler to support $LDL^T$, rank update/downdates, use in nonlinear programming and in computer graphics/machine learning applications.

- **MatRox**, a framework that generates code for approximate matrix computations, i.e. hierarchal matrix methods.

- ParSy’s source code is publicly available from: 