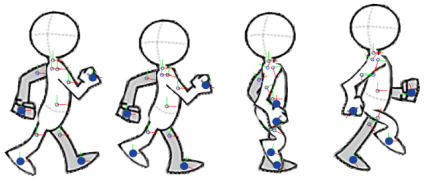
Sympiler: Inspection and Transformation of Sparse Matrix Computations

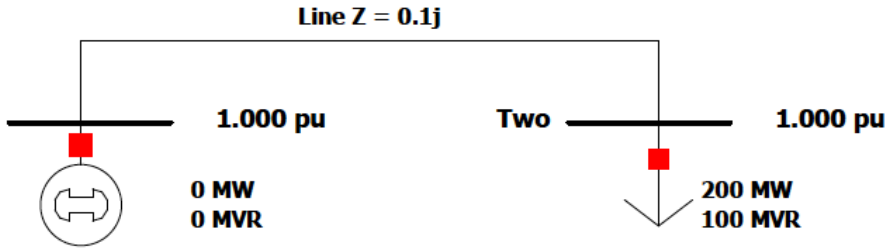
Maryam Mehri Dehnavi

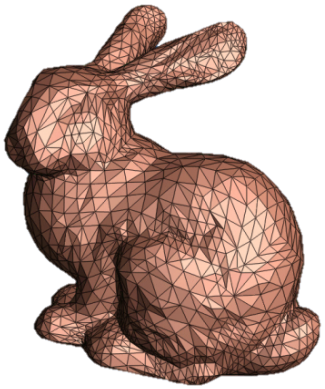
University of Toronto

Symbolic Analysis in Sparse Matrix Solvers

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*.



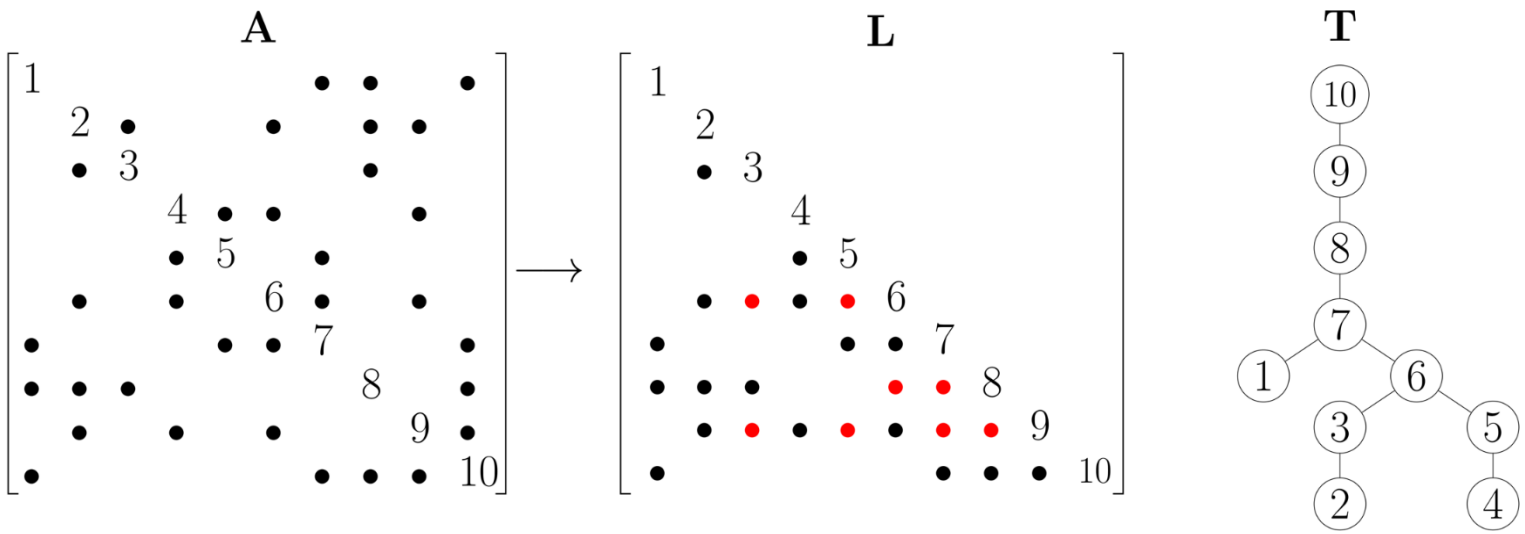




EXAMPLE: CHOLESKY FACTORIZATION

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

- **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.

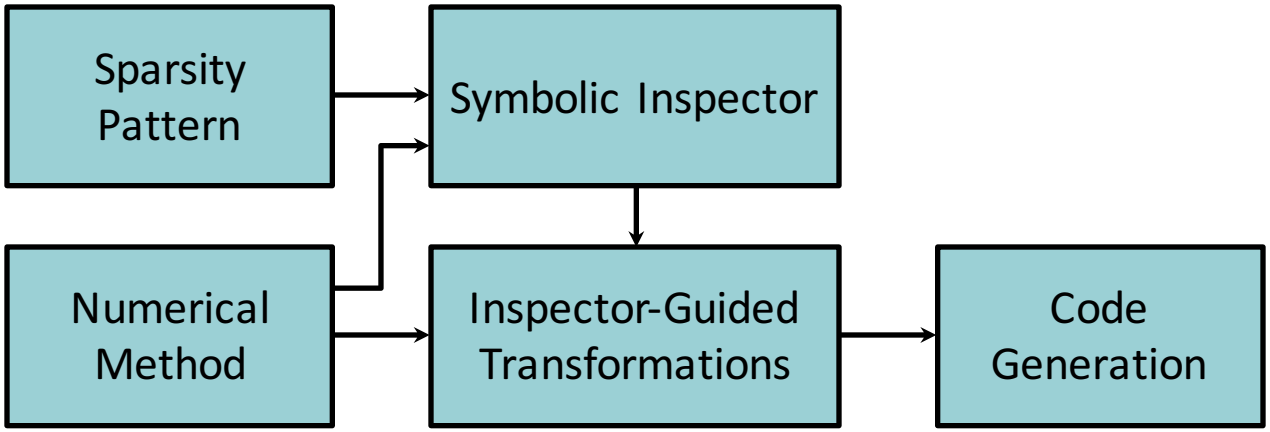
**Cheshmi, Kamil, Strout, Dehnavi "Sympiler: transforming sparse matrix codes by decoupling symbolic analysis." SC17*

ParSy (Parallel Sympiler)

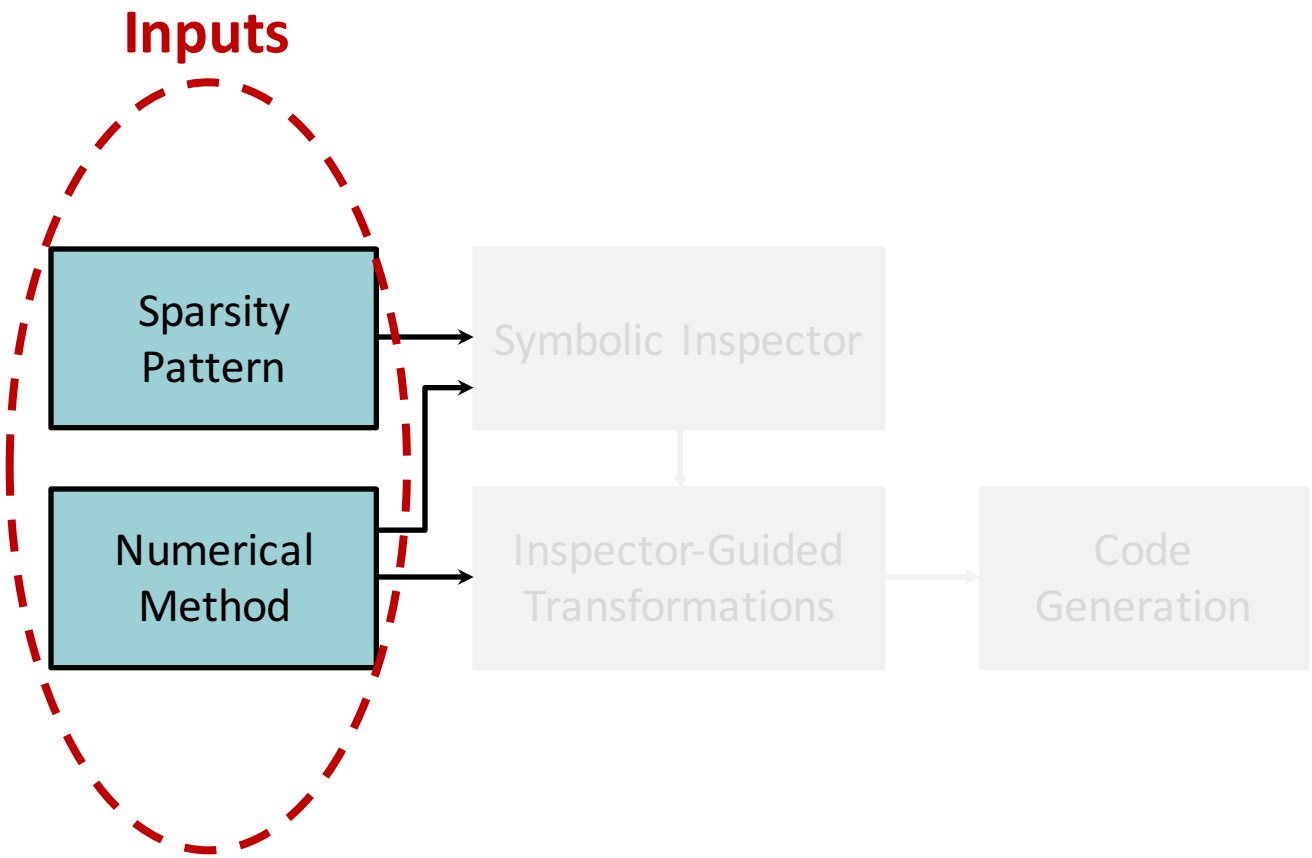
- **Sympiler** is a domain-specific compiler for generating high-performance code for sparse solvers.
  - It uses symbolic information to transform the sparse code.
  - Sympiler applies prune and block transformations but does not support parallelism for multicore.
- ParSy* generates parallel code for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

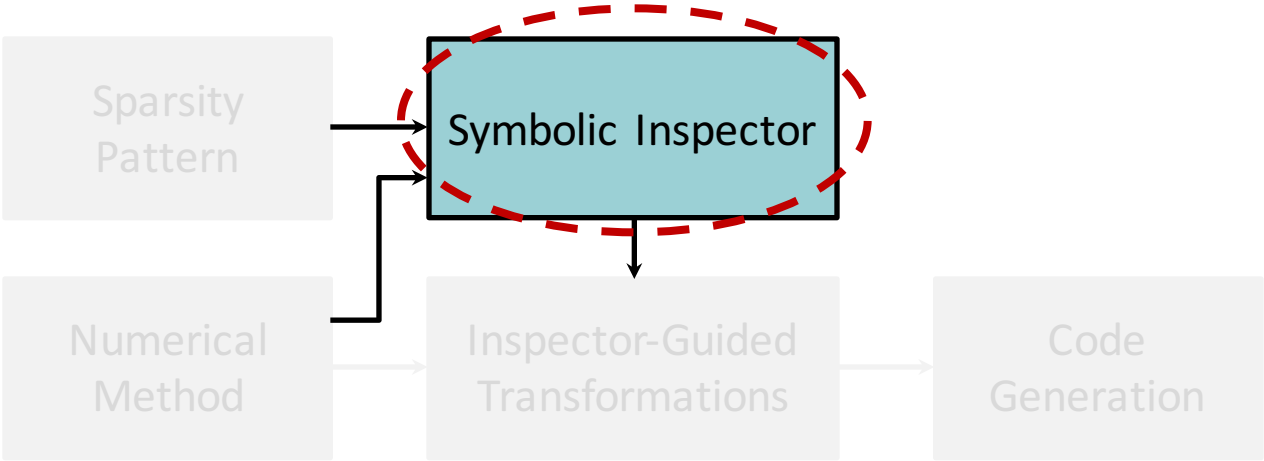


ParSy (and Sympiler) Internals

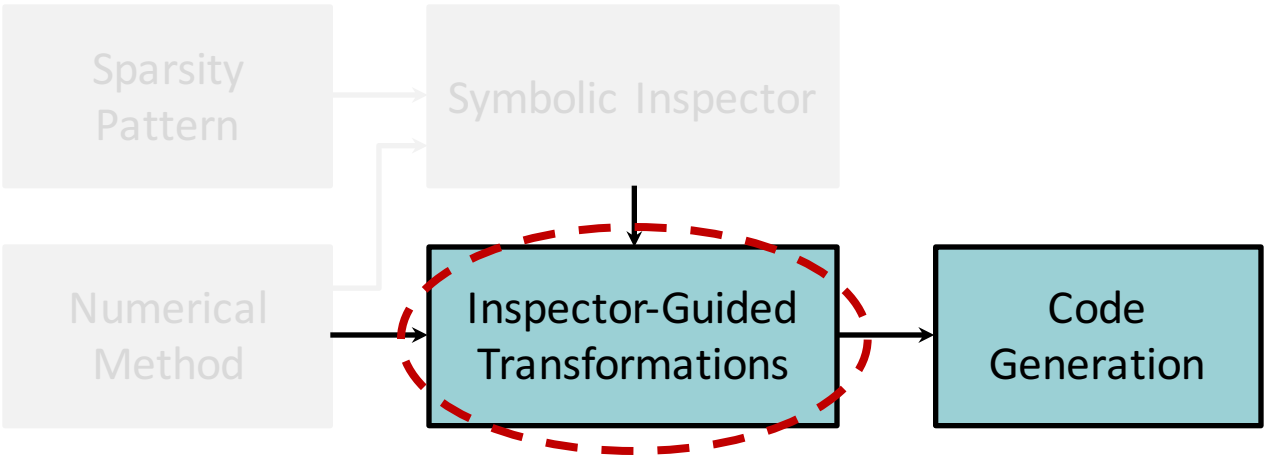


ParSy (and Sympiler) Internals

Inspection of the dependence graph to create *inspection sets*

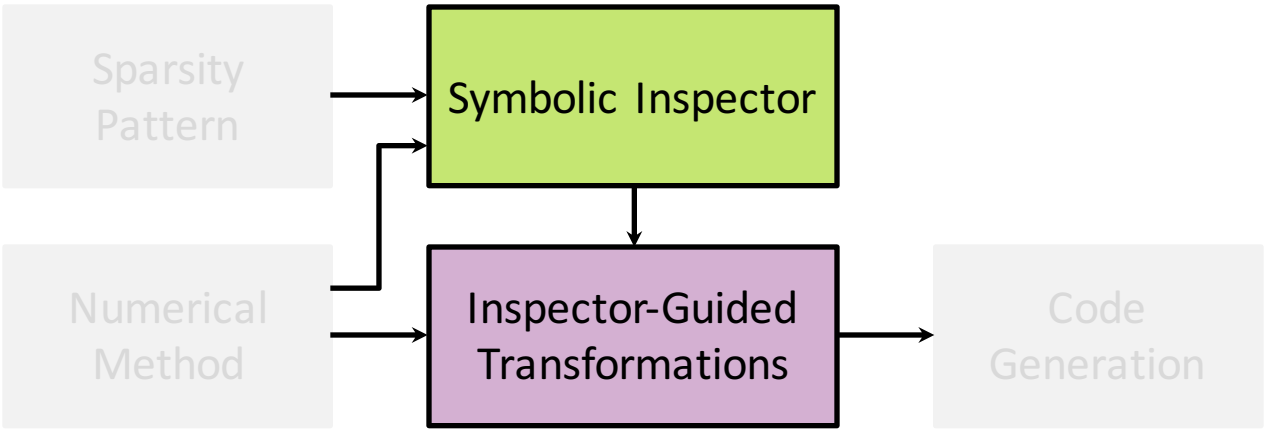


ParSy (and Sympiler) Internals



The *inspection sets* are used to transform the sparse code

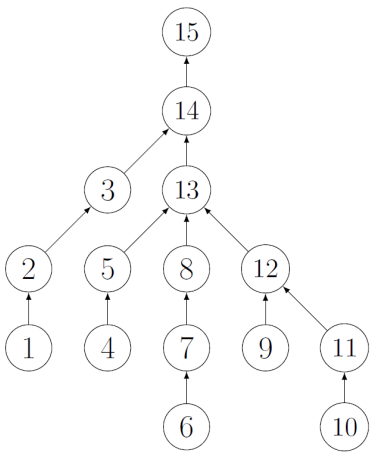
ParSy (and Sympiler) Internals

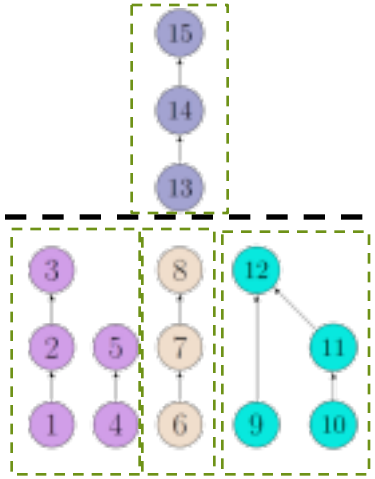


Example: ParSy introduces **H-Level inspection** and **H-Level transformation** to generate parallel code.

H-Level Inspection in Cholesky

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**.





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

H-Level Transformations

Hierarchical Level (H-Level) transformation transforms ParSy's internally annotated code using the H-Level set to generate parallel code.

```
H-Level:
for(I1) {
    …
    for(In(I1)) {
Atomic:
        c /= a[idx(I1,...,In)];
    }
}
```

Internally annotated code

H-Level set

```
for ( every l−partition l) {
#pragma omp parallel for private(pVars)
   for ( every w−partition w ) {
      for ( every v ∈ HLevelSet[l][w] ) {
         I1 = v
         ...
            for(In(I1)) {
#pragma omp atomic
               c /= a[idx(I1,...,In)];
            }
}}}
```

Transformed with H-Level

The ParSy-Generated Code for Cholesky

```
H-Level:
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));
}}
```

Internally annotated code

H-Level set

```
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]);
```

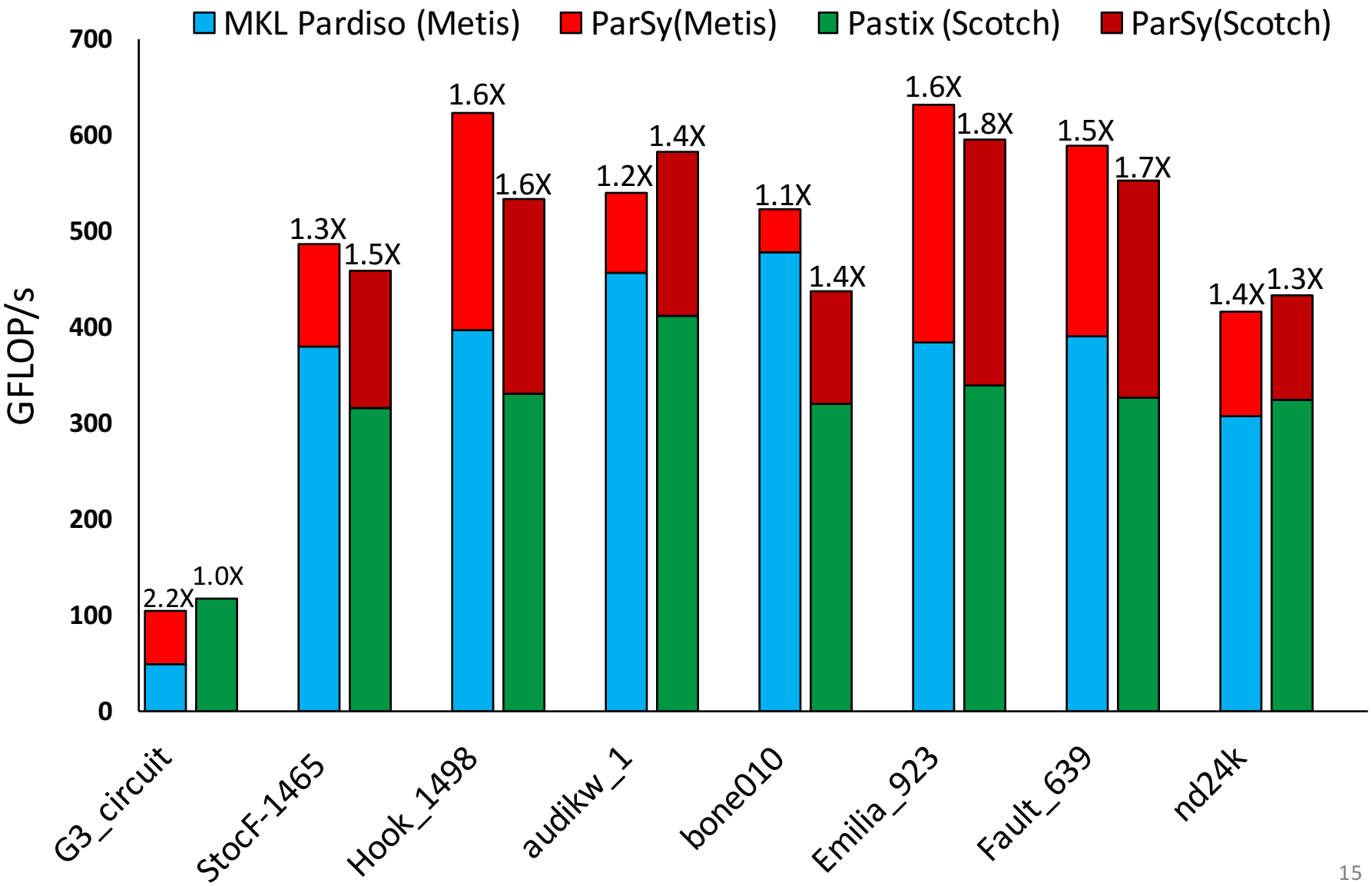
Transformed with H-Level

Experimental Setup

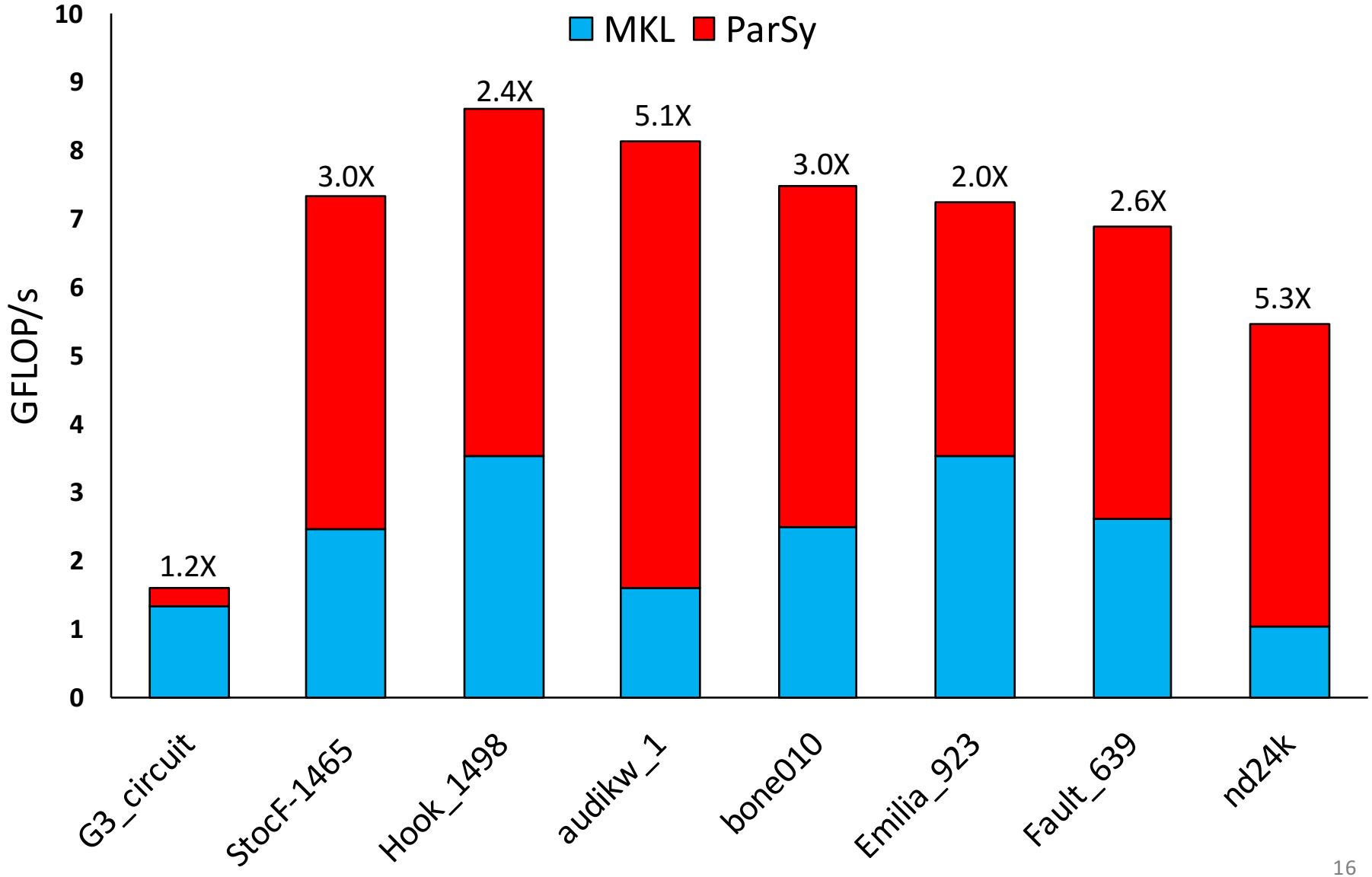
Numeric and symbolic times are compared separately where applicable; **Target processor**: Intel® Xeon® Platinum 8160 (Skylake); **Benchmarks**: Suitesparse matrix collection

Name	Application	Order (10^3)	Non-zeros (10^6)
G3_circuit	Circuit simulation	1585	127.3
StocF_1465	Computational fluid dynamics problem	1465.1	1245
Hook_1498	3D mechanical problem	1498	1783.8
audikw_1	Structural problem	943.7	1473.1
bone010	Model reduction problem	986.8	1210.1
Emilia_923	Geomechanical model	923.1	1992
Fault_639	Contact mechanics	638.8	1275.4
nd24k	2D/3D problem	72	435.9

ParSy vs Libraries: Cholesky



ParSy vs Libraries: Triangular Solve



Inspection Overhead: Cholesky

Name	MKL Acc Time / ParSy Acc Time	Pastix Acc Time / ParSy Acc Time
G3_circuit	**1.4**	0.9
StocF_1465	**1.4**	0.9
Hook_1498	**1.25**	**1**
audikw_1	**1.25**	**1**
bone010	**1.11**	0.9
Emilia_923	**1.4**	**1.25**
Fault_639	**1.25**	**1.25**
nd24k	**1.25**	**1**

Acc Time = Symbolic analysis time + Numerical factorization

On-Going Work

- 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:

https://www.Sympiler.com/