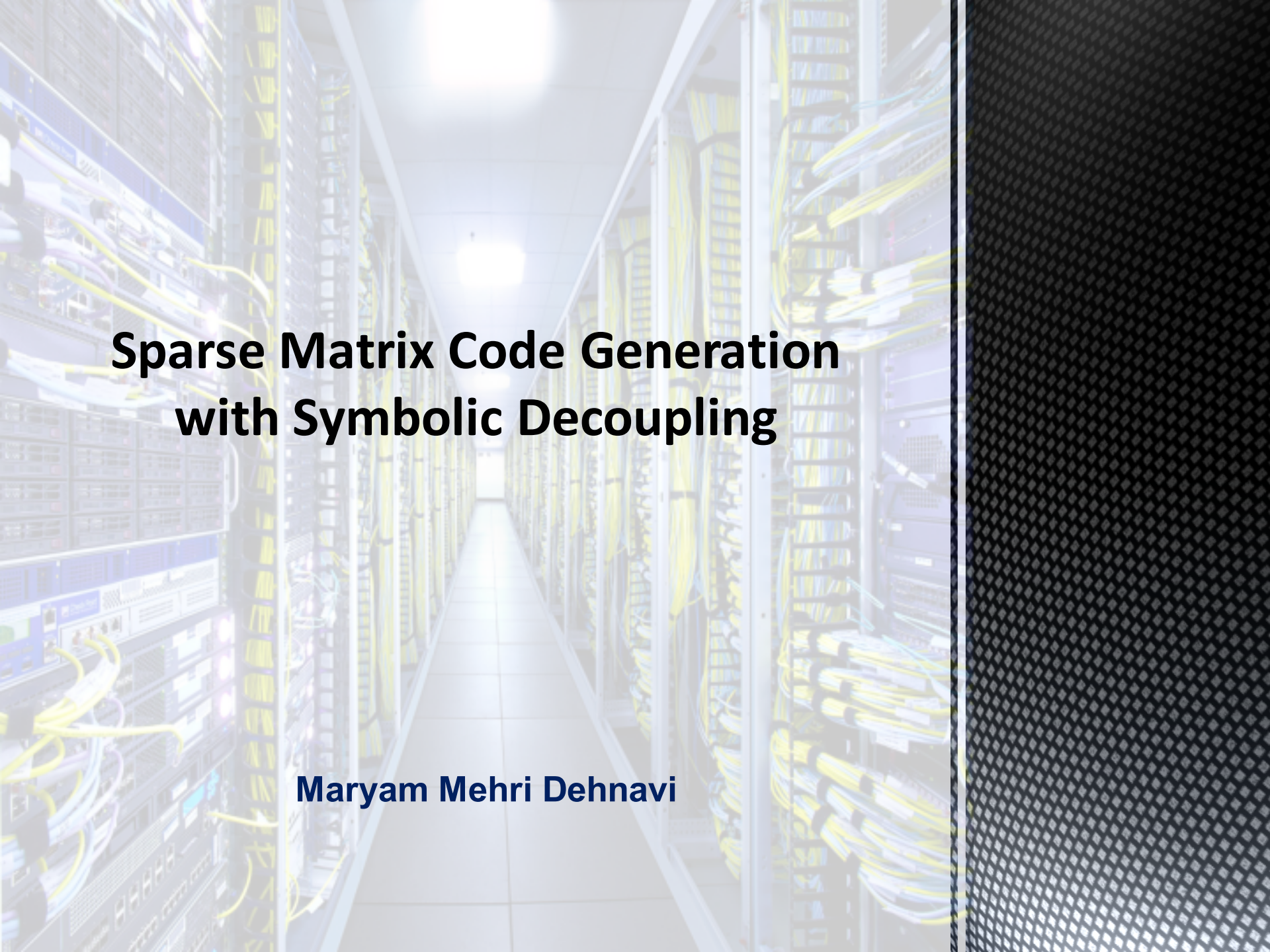


Sparse Matrix Code Generation with Symbolic Decoupling

Maryam Mehri Dehnavi

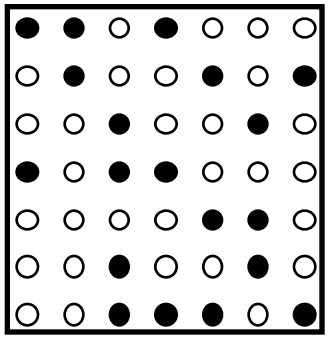
Outline

- Overview
- How Sympiler transforms computation and communication patterns
- Sympiler internals
- ParSy: Parallel Sympiler
- Results

Linear Systems of Equations

Many simulations in scientific problems require solving a **linear system of equations**, $Ax = b$.

NXN matrix often sparse



Sparse A

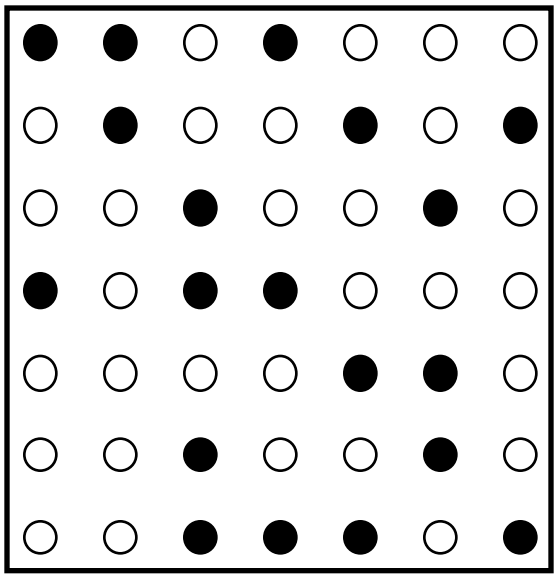
NX1 column vector of unknown

Represents whatever is "driving" the problem, e.g. boundary conditions, remote sources, etc.

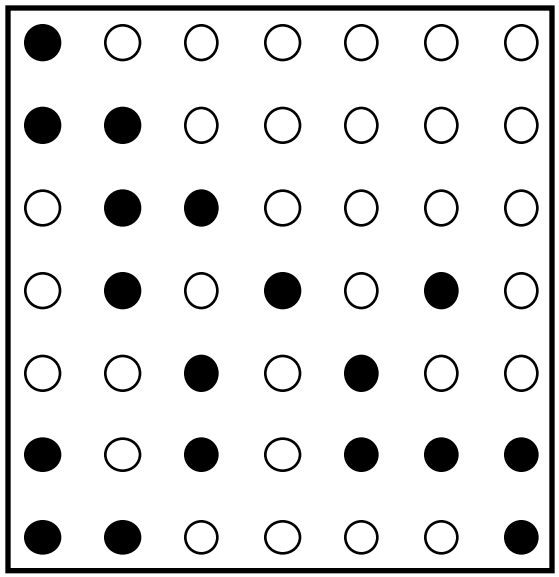
Sparsity Pattern and Symbolic Information

The pattern and location of non-zeros in a sparse matrix is called the **sparsity pattern**.

Information about this pattern independent of the non-zero values is called **symbolic information**.



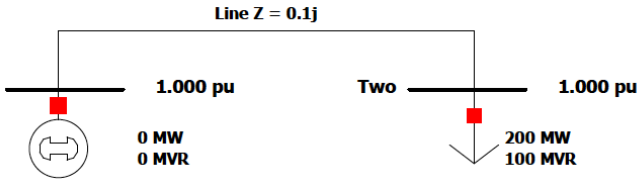
Sparse A

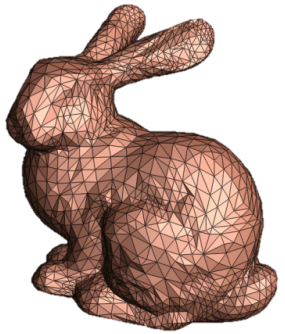


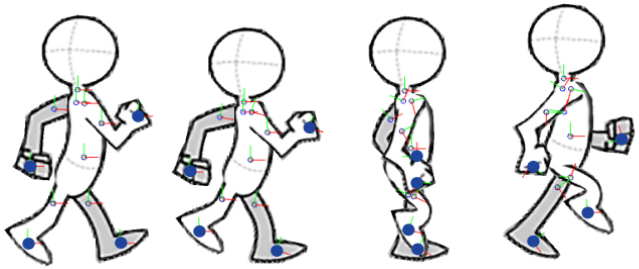
Sparse B

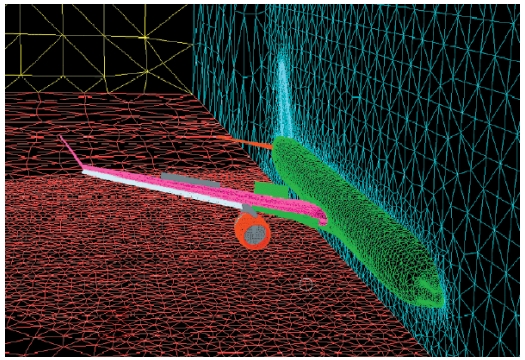
Static Sparsity Patterns

The **sparsity patterns** in many applications such as power modeling and circuit simulation often remain **static** for a period of time since the structure arises from the physical domain and the governing equations.









Implementation of ParSy

- **Sympiler*** generates high-performance code for sparse solvers with symbolic decoupling.
  - It uses symbolic information to transform the sparse code.
  - Sympiler 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.

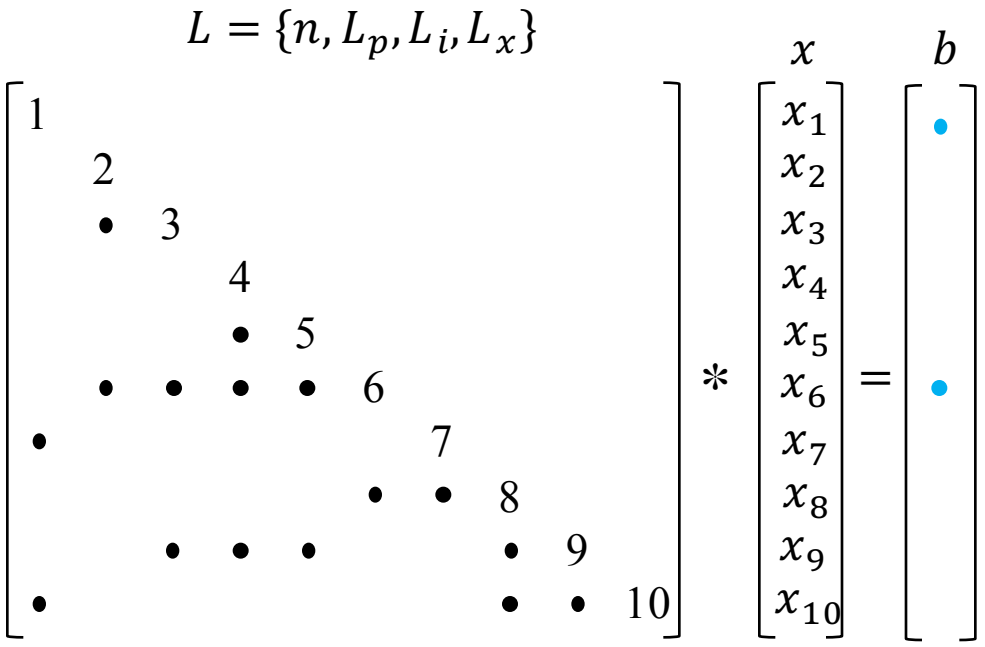
Cheshmi, Kamil, Strout, Dehnavi, Sympiler and ParSy, SC17, SC18

OUTLINE

- Overview
- How Sympiler transforms computation and communication patterns
- Sympiler internals
- ParSy: Parallel Sympiler
- Results

EXAMPLE: A SPARSE TRIANGULAR SYSTEM SOLVER

Find the solution to x, $Lx = b$ where L is sparse lower triangular matrix.



Solving a Sparse Triangular System

Solve *Lx=b* with *L* unit lower triangular; *L, x, b* are sparse.

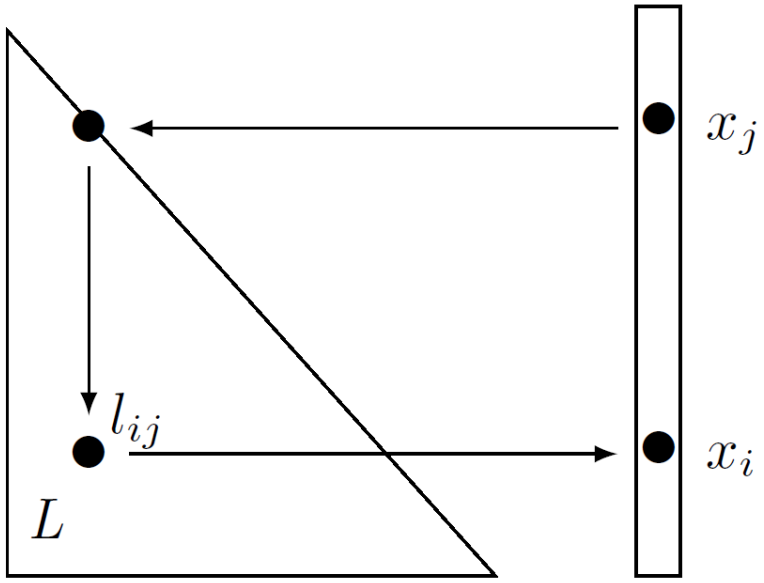
if $b_i \neq 0$ then $x_i \neq 0$

if $x_j \neq 0$ and $\exists i \left(l_{ij} \neq 0 \right)$ then $x_i \neq 0$

start with pattern β of b

graph L: edge (j, i) if $l_{ij} \neq 0$

$\chi = Reach_L(\beta)$



Symbolic Analysis in the Sparse Triangular System Solver

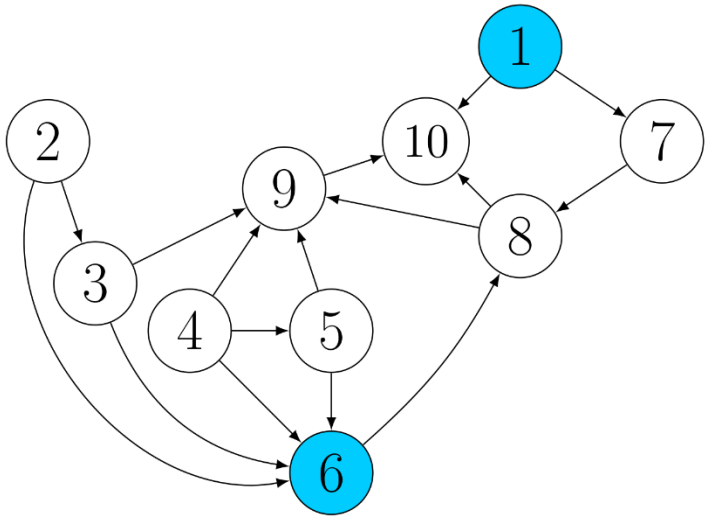
Depth First Search (DFS)

Dependence Graph (DG_L)

$$L = \{n, L_p, L_i, L_x\}$$

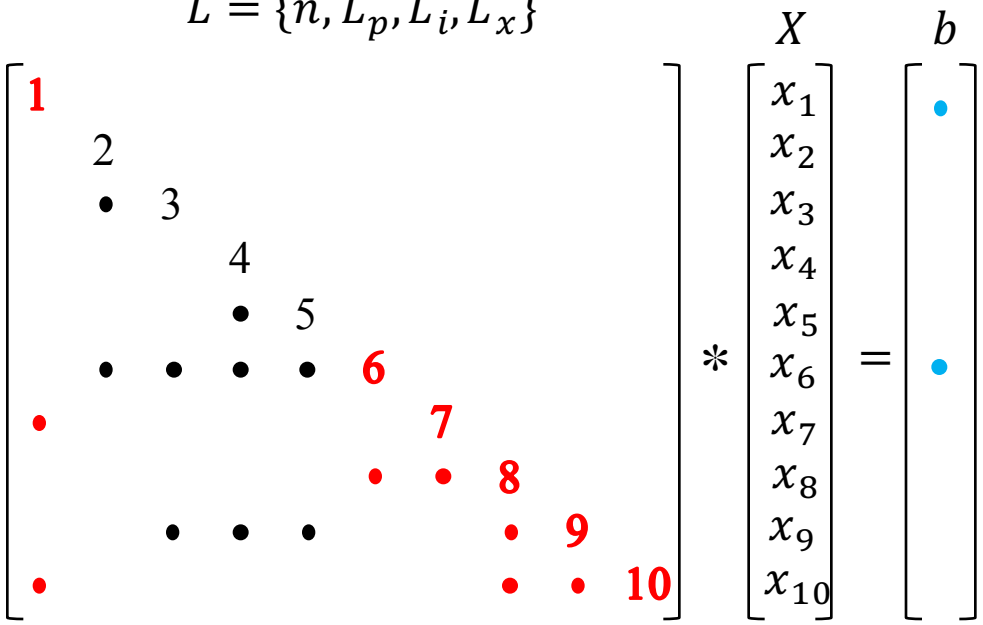
$$\begin{bmatrix} 1 & & & & & & & & & \\ & 2 & & & & & & & & \\ \bullet & & 3 & & & & & & & \\ & & & 4 & & & & & & \\ & & & \bullet & 5 & & & & & \\ \bullet & \bullet & & \bullet & \bullet & 6 & & & & \\ \bullet & & & & & & 7 & & & \\ & & & & & \bullet & \bullet & 8 & & \\ & \bullet & & \bullet & \bullet & & & \bullet & 9 & \\ \bullet & & & & & & & \bullet & \bullet & 10 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} \bullet \\ \\ \\ \\ \\ \bullet \\ \\ \\ \\ \end{bmatrix}$$

X b



Symbolic Analysis in the Sparse Triangular System Solver

$L = \{n, L_p, L_i, L_x\}$

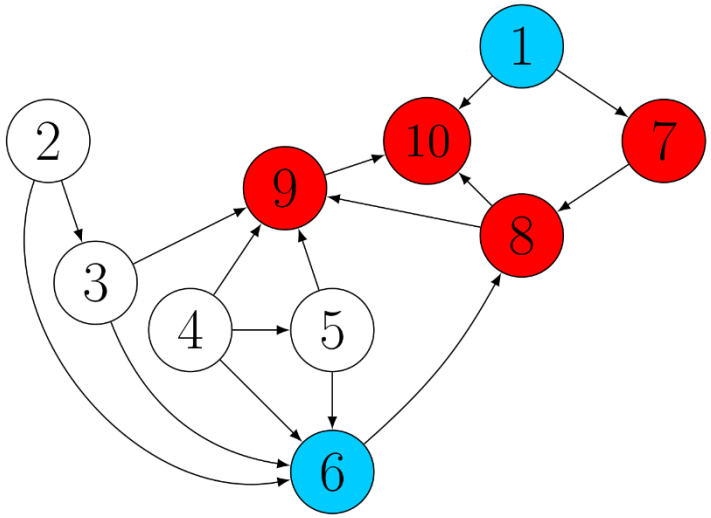


Depth First Search (DFS)

Dependence Graph (DG_L)

$\beta = \{i | b_i \neq 0\} = \{1,6\}$

$Reach_L(\beta) = \{1,6,7,8,9,10\}$



Transforming the Sparse Code

Standard Code

```
for (j=0; j<n; j++){

  x[j]/=Lx[Lp[j]];

  for (p=Lp[j]+1; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
 }
```

Transforming the Sparse Code

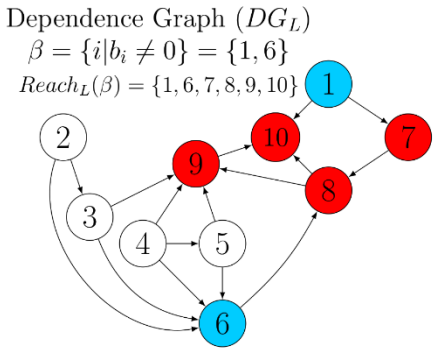
Standard Code

```
for (j=0; j<n; j++){

  x[j]/=Lx[Lp[j]];

  for (p=Lp[j]+1; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
 }
```

Prune the iteration space:
Reachset = {6,1,7,8,9,10}

Symbolically-Guided Code

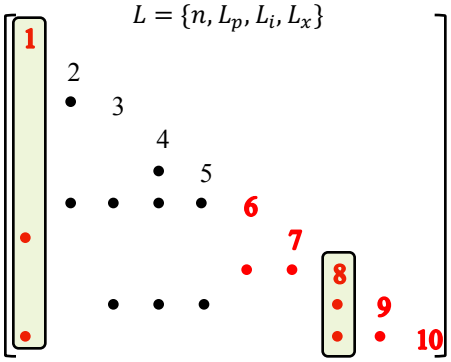
```
for (px=0; px<RSsize; px++) {
 j=reachset[px];          // = 6
 x[j]/=Lx[Lp[j]]
 p=Lp[j]+1;
 for (; p<Lp[j+1]; p++)
   x[Li[p]]-=Lx[p]*x[j];
 }
```



Transforming the Sparse Code

Symbolically-Guided Code

```
for (px=0; px<RSsize; px++) {
  j=reachset[px];
  x[j]/=Lx[Lp[j]]
  p=Lp[j]+1;
  for (; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
  }
```

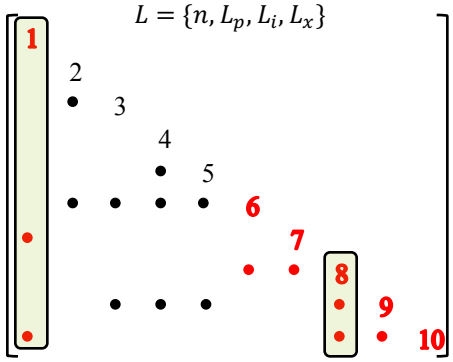


Transforming the Sparse Code

Symbolically-Guided Code

```
for (px=0; px<RSsize; px++) {
  j=reachset[px];
  x[j]/=Lx[Lp[j]]
  p=Lp[j]+1;
  for (; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
  }
```

Peel columns with non-zeros ≥ 2



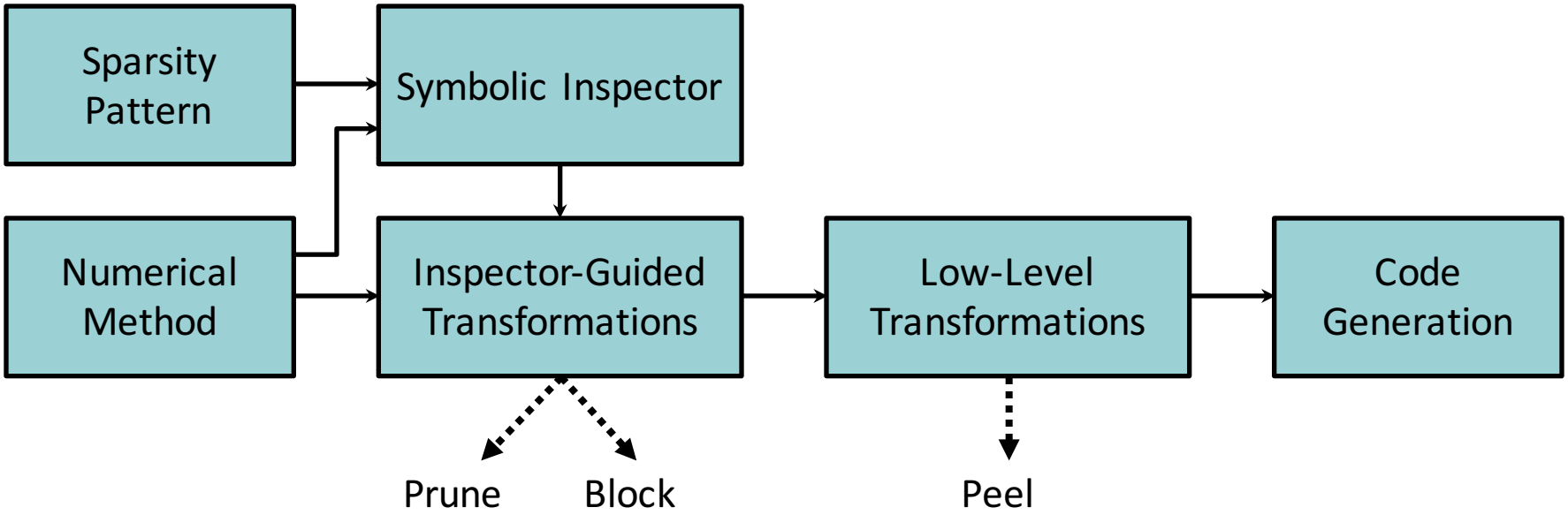
Optimized Code

```
x=b;
x[0] /= Lx[0]; // Peel col 1
for(p = 1; p < 3; p++)
    x[Li[p]] -= Lx[p] * x[0];

for(px=1;px<3;px++){
j=reachSet[px];x[j]/=Lx[Lp[j]];
  for(p=Lp[j]+1;p<Lp[j+1];p++)
      x[Li[p]]-=Lx[p]*x[j];}

x[7] /= Lx[20]; // Peel col 8
for(p = 21; p < 23; p++)
    x[Li[p]] -= Lx[p] * x[7];

for(px=4;px<reachSetSize;px++){
j=reachSet[px];x[j]/=Lx[Lp[j]];
  for(p=Lp[j]+1;p<Lp[j+1];p++)
      x[Li[p]]-=Lx[p]*x[j];
}
```

OUTLINE

- Overview
- How Sympiler transforms computation computation and communication patterns
- Sympiler internals
- ParSy: Parallel Sympiler
- Results

SYMPILER INTERNALS



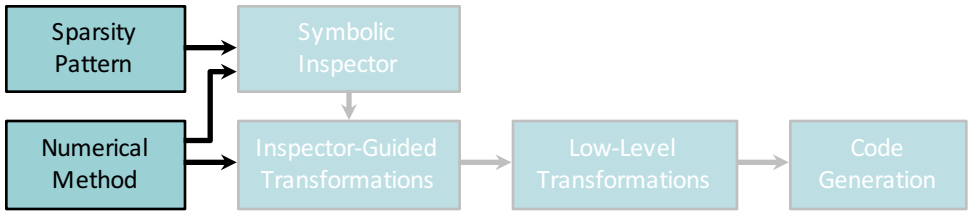
The Embedded Domain-Specific Language

The user will represent their **numerical method**, the **sparsity patterns**, updates to the sparsity using the DSL.

```
int main() {
 Sparse L(Float(64),"L.mtx");
 Sparse b(Float(64),"b.mtx");

 Triangular trns(L,b);
 trns.sympile_to_c("triang");
}
```

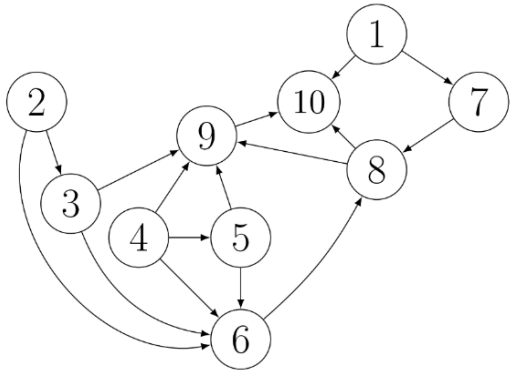
Input sparsity patterns

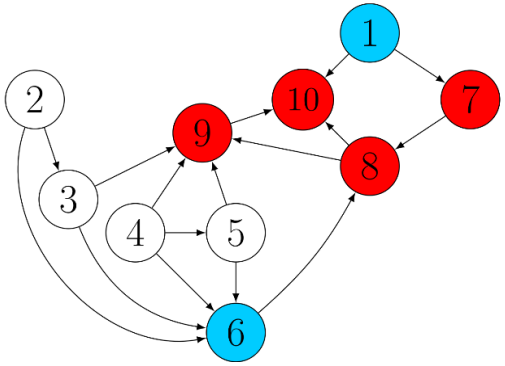
Numerical method



FINDING THE REACHSET WITH SYMBOLIC INSPECTION

The symbolic inspector creates an **inspection graph** and traverses it with an **inspection strategy**. The result of the inspection is an **inspection set**.





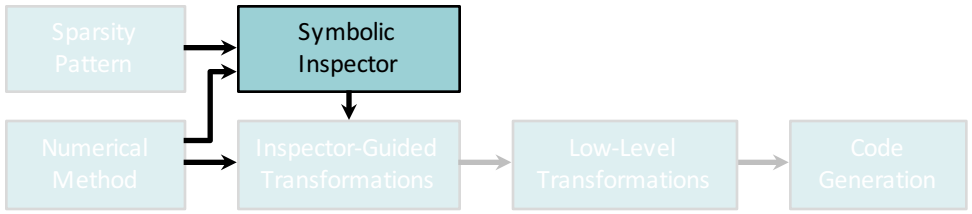
$$Reach_L(\beta) = \{1,6,7,9,10\}$$

Inspection Graph: DG_L

Inspection Strategy: DFS

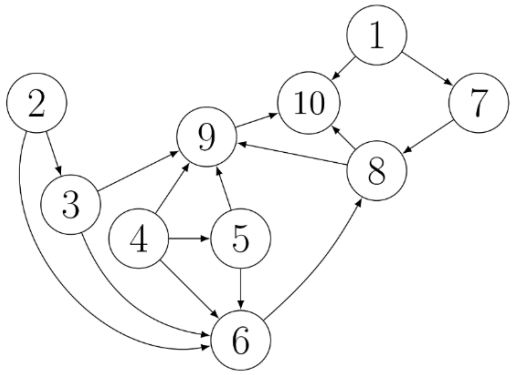
Inspection Set: Reachset

Will guide **Prune** optimizations

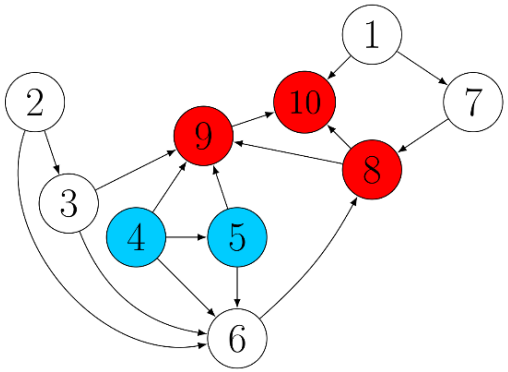


Finding the Blockset with Symbolic Inspection

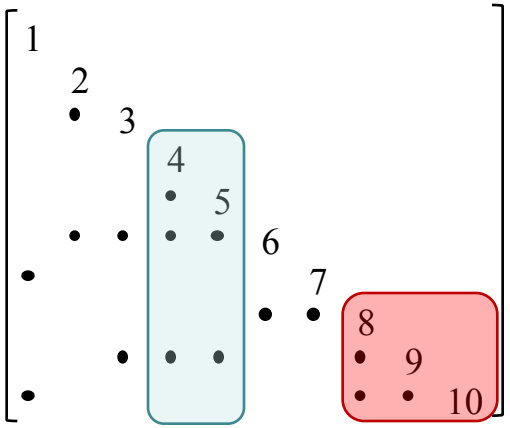
Inspection for finding blocks of columns with similar computation patterns.



Inspection Graph:
DG_L

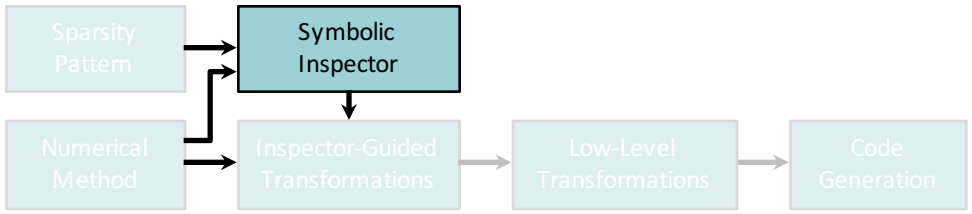


Inspection Strategy:
Node equivalence



Inspection Set:
Blockset

Will guide **Block** optimizations

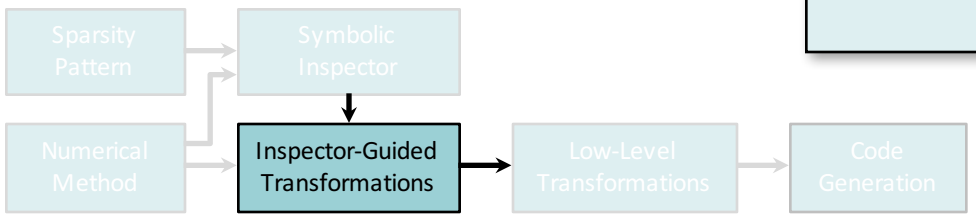


The Initial Abstract Syntax Tree (AST) with Annotations

```
int main() {
 Sparse L(Float(64),"L.mtx");
 Sparse b(Float(64),"b.mtx");

 Triangular trns(L,b);
 trns.sympile_to_c("triang");
}
```

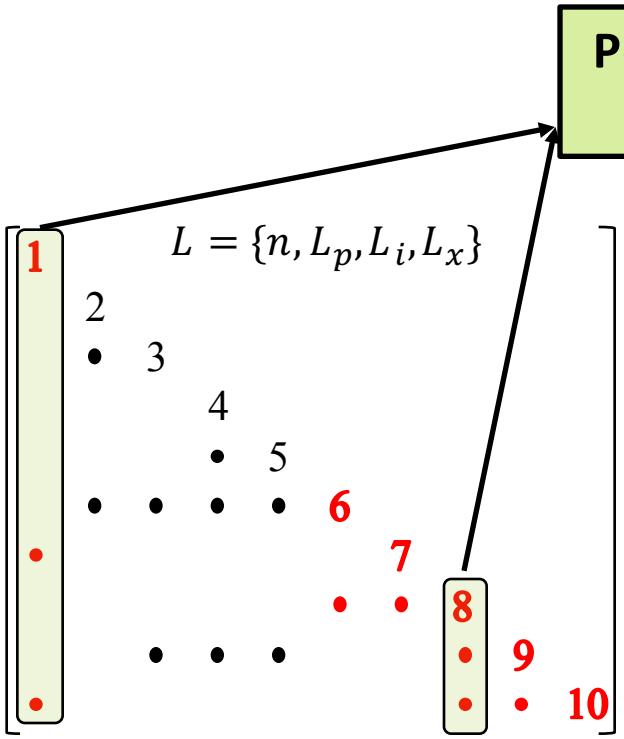
```
Prune
peel(0,7)
for sol.j0 in 0..Lsp.n
 Block
 x[bsp_j0] /= Lx[Lsp.diag_j0)];
  Block
    for sol.j1 in Lsp.col_j0..Lsp.col_j0+1
    x[Lsp.row_j1] -= Lx[j1]*x[bsp_j0];
```

Annotations for **inspector-guided transformations**

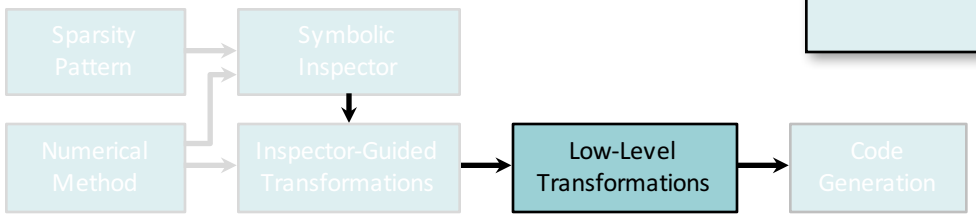


AUTOTUNING OPTIMIZATION PARAMETERS AND TRANSFORMATIONS

The decision to apply transformations and their thresholds are determined with **autotuning**.




```
Prune
peel(0,7) //peel columns 1 and 8
for sol.j₀ in 0..Lsp.n
 Block
x[bsp_j0] /= Lx[Lsp.diag_j0)];
    Block
    for sol.j₁ in Lsp.col_j0..Lsp.col_j0+1
   x[Lsp.row_j1] -= Lx[j₁]*x[bsp_j0];
```



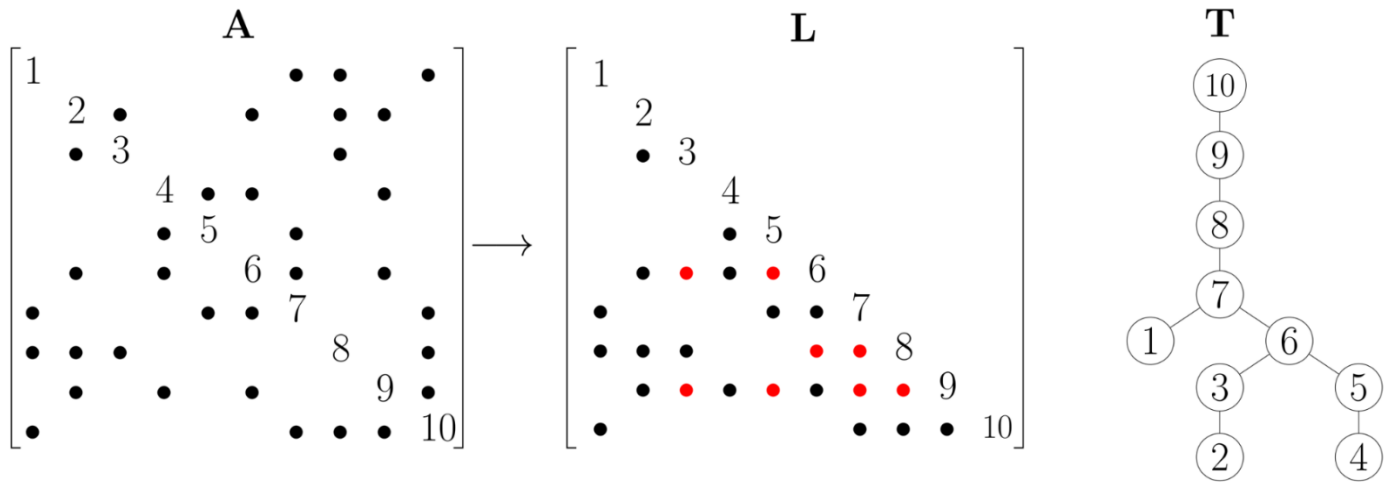
OUTLINE

- Overview
- How Sympiler transforms computation and communication patterns
- Sympiler internals
- ParSy: Parallel Sympiler
- Results

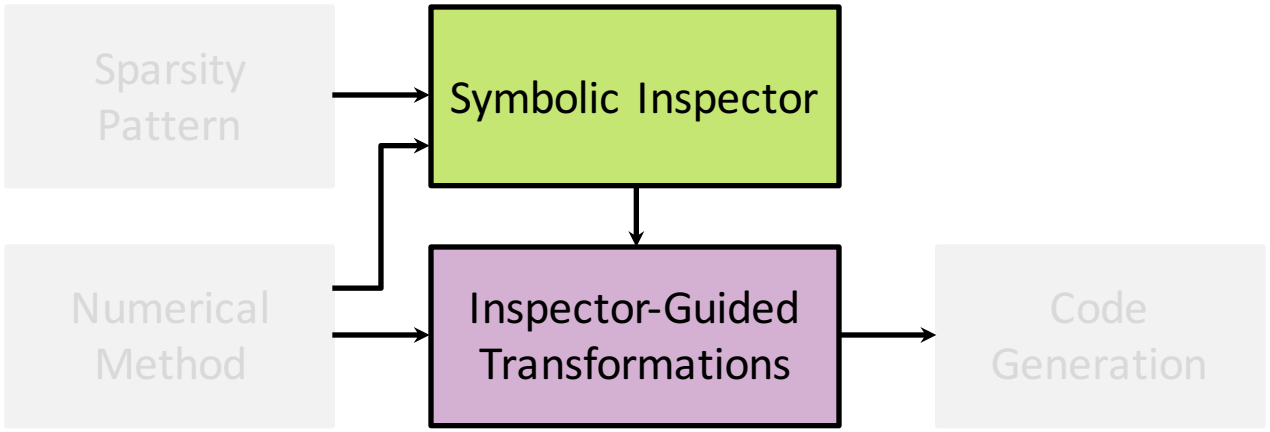
CHOLESKY FACTORIZATION

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.



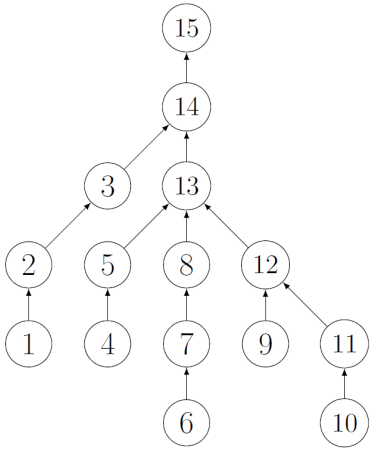
ParSy (and Sympiler) Internals

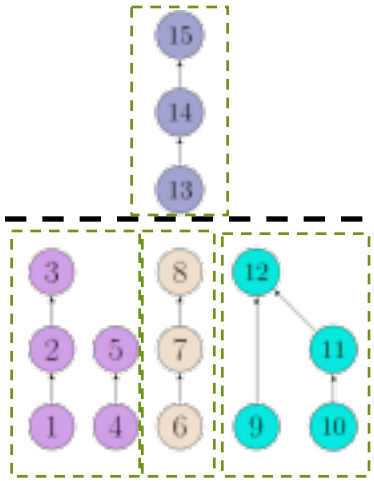


ParSy introduces **H-Level inspection** and **H-Level transformation** to generate parallel code.

H-Level Inspection

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

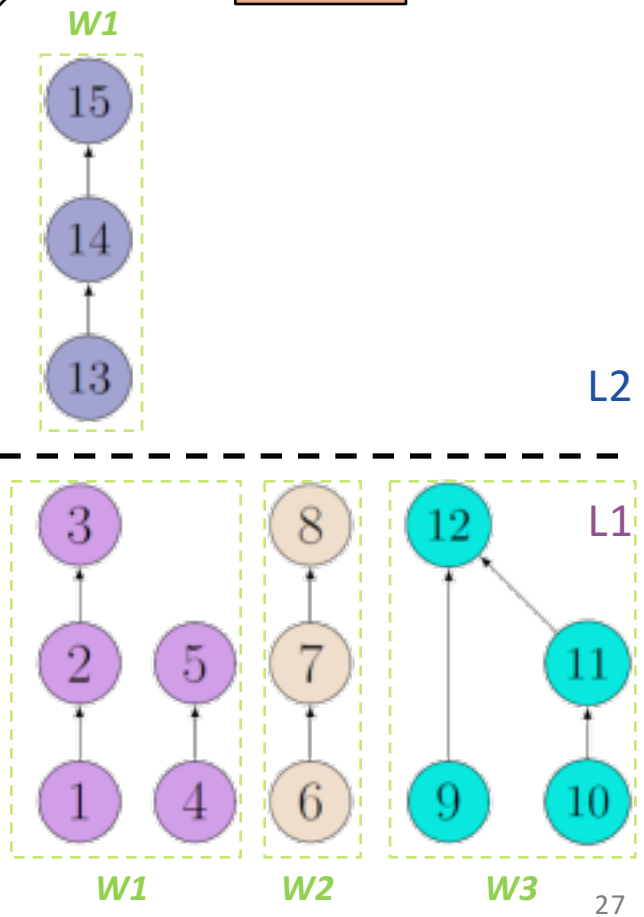
The Load-Balanced Level Coarsening (LBC) Algorithm

Step 1: L-partitioning
- Find the initial cut: first l-partition
- Build the rest of the l-partitions

Step 2: W-partitioning
- A cost model for load balance

Step 3: Reordering
- Reduce intra-partition cost.

LBC



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

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

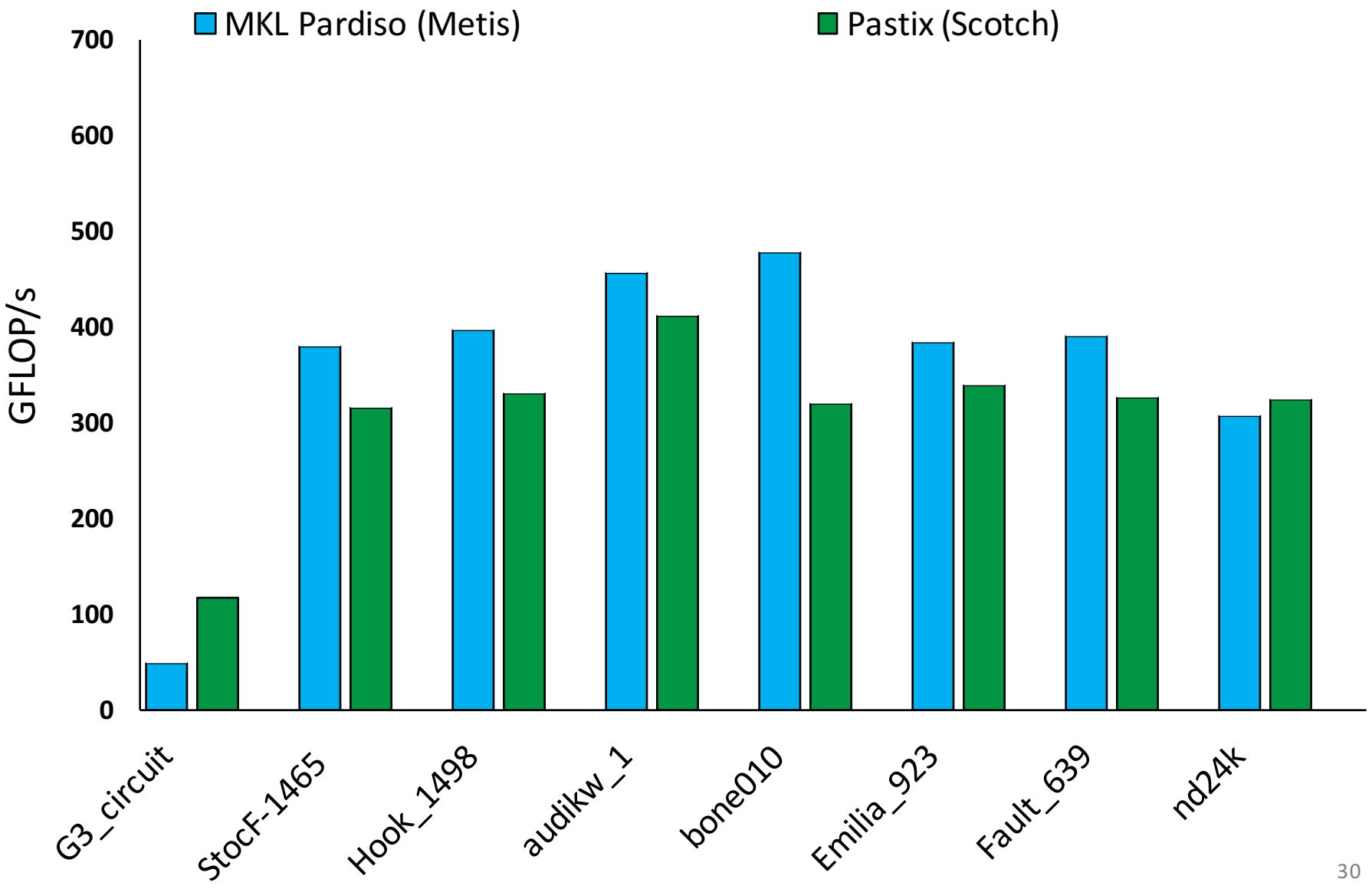
```
for(every l-partition l < nlevels-1){
#pragma omp parallel for private(f){
  for(every w-partition w){
    for(every v ∈ HLevelSet[l][w]){
      int i = v;
      b1 = block2col[i];b2 = block2col[i+1];
      f = A(:,b1:b2);
      for(block r=0 to i-1 L(i,r)!=0){
        f-=GEMM(L(b1:n,r),transpose(L(i,r)));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

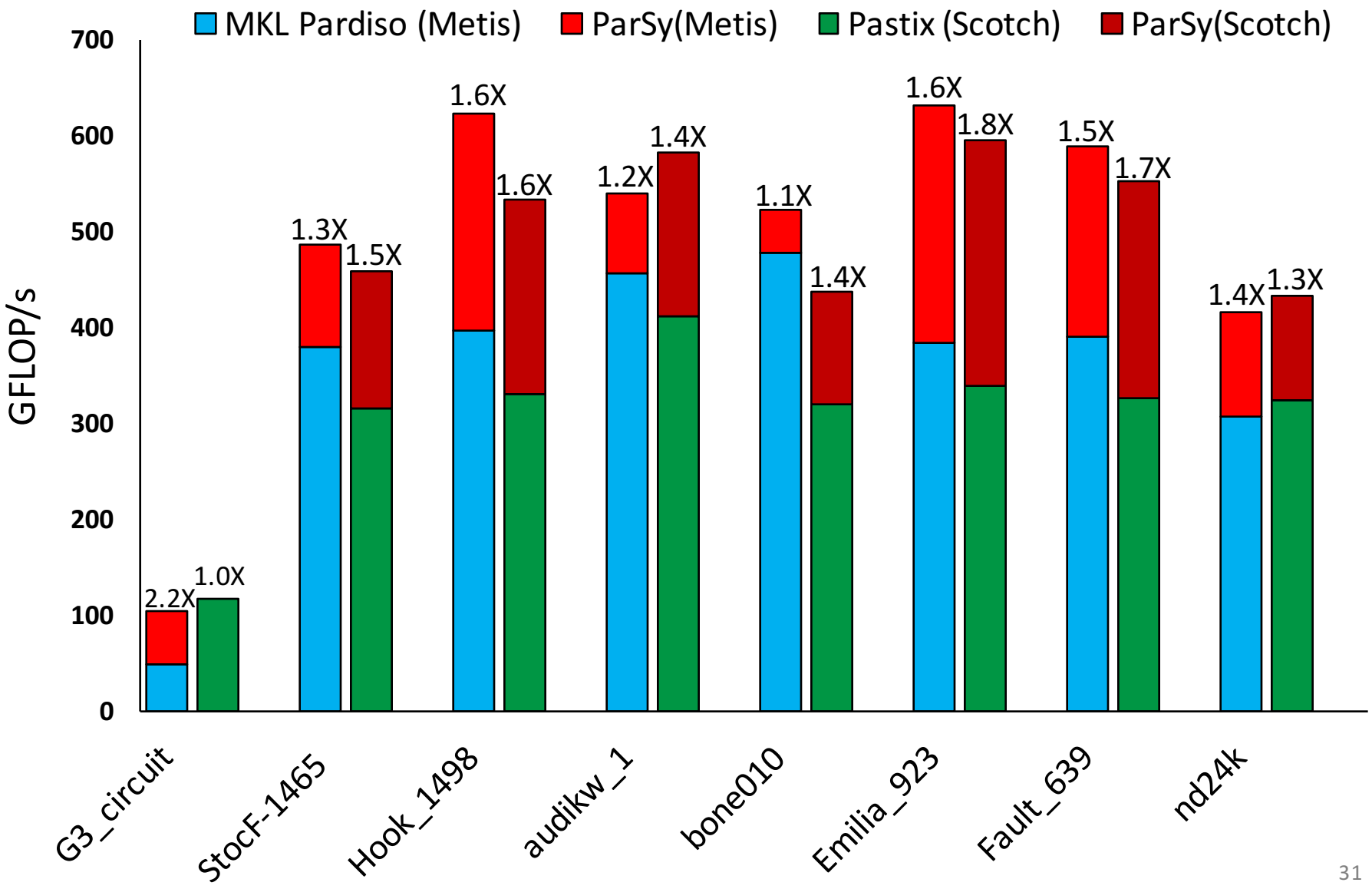
Outline

- Overview
- How Sympiler transforms computation and communication patterns
- Sympiler internals
- ParSy: Parallel Sympiler
- Results

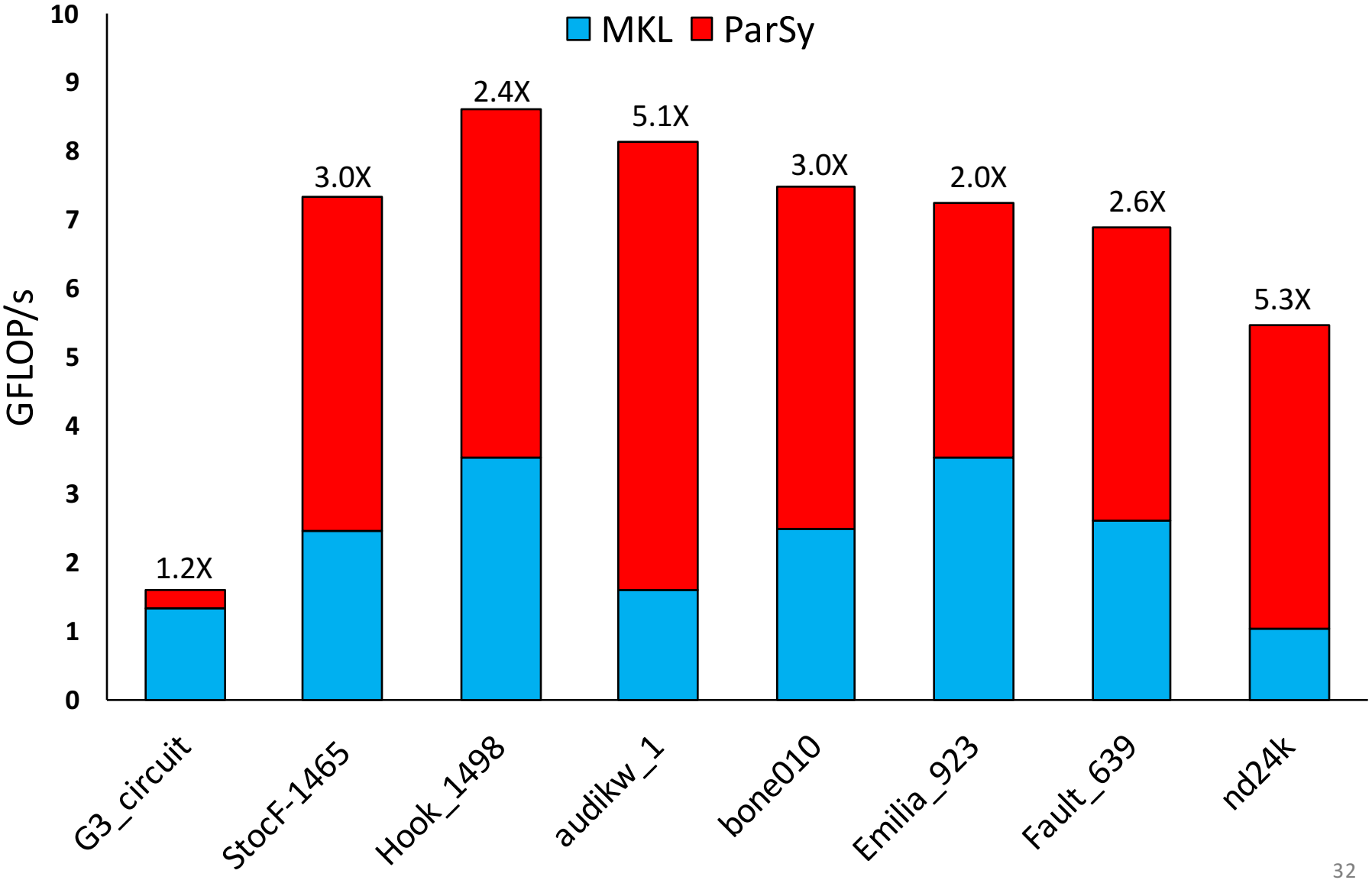
PARSY VS LIBRARIES: CHOLESKY



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.
- Extending ParSy/Sympiler to support LDL^T, rank update/downdates, use in nonlinear programming, and in computer graphics/machine learning applications.
- MatRox is a framework that we are building on top of Sympiler to support approximate matrix computations, i.e. hierarchal matrix methods
- ParSy's source code is publicly available from:

https://www.Sympiler.com/