# ParSy: Inspection and Transformation of Sparse Matrix Computations for Parallelism

**Kazem Cheshmi [1], Shoaib Kamil [2],**

**Michelle Strout [3], Maryam Mehri Dehnavi [1]**

University of Toronto[1], Adobe Research[2], University of Arizona[3]

# Outline

- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

# Outline

- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

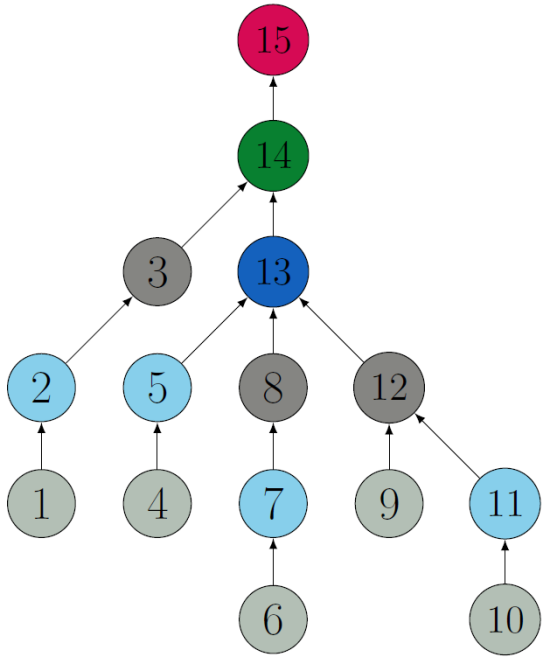
# Compiler Loop Parallelism in Sparse Kernels

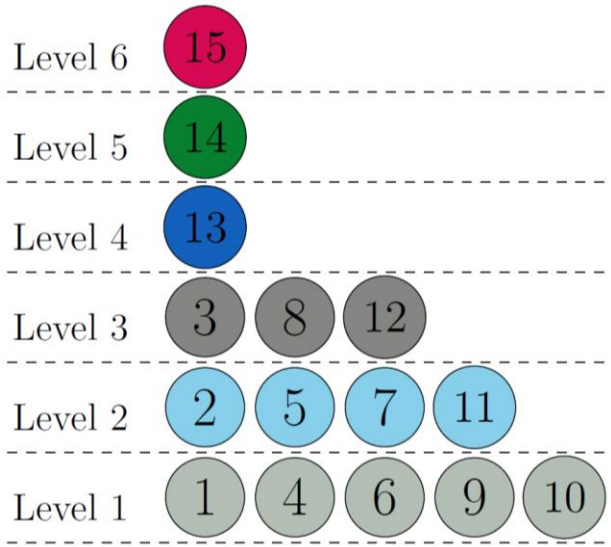
- Loop-carried dependences widely exist in sparse kernels such as the sparse matrix factorization.
- Dependency analysis is difficult in sparse kernels because of indirect memory accesses e.g., A[B[i]].
- Sparse polyhedral frameworks* typically use **Wavefront parallelism**.

*Strout, Hall, and Olschanowsky "The Sparse Polyhedral Framework: Composing Compiler-Generated Inspector-Executor Code." *Proceedings of the IEEE* 99 (2018): 1-15.

# Loop Parallelism in Wavefront Methods

Wavefront parallelism for sparse codes might suffer from load imbalance and reduced locality.

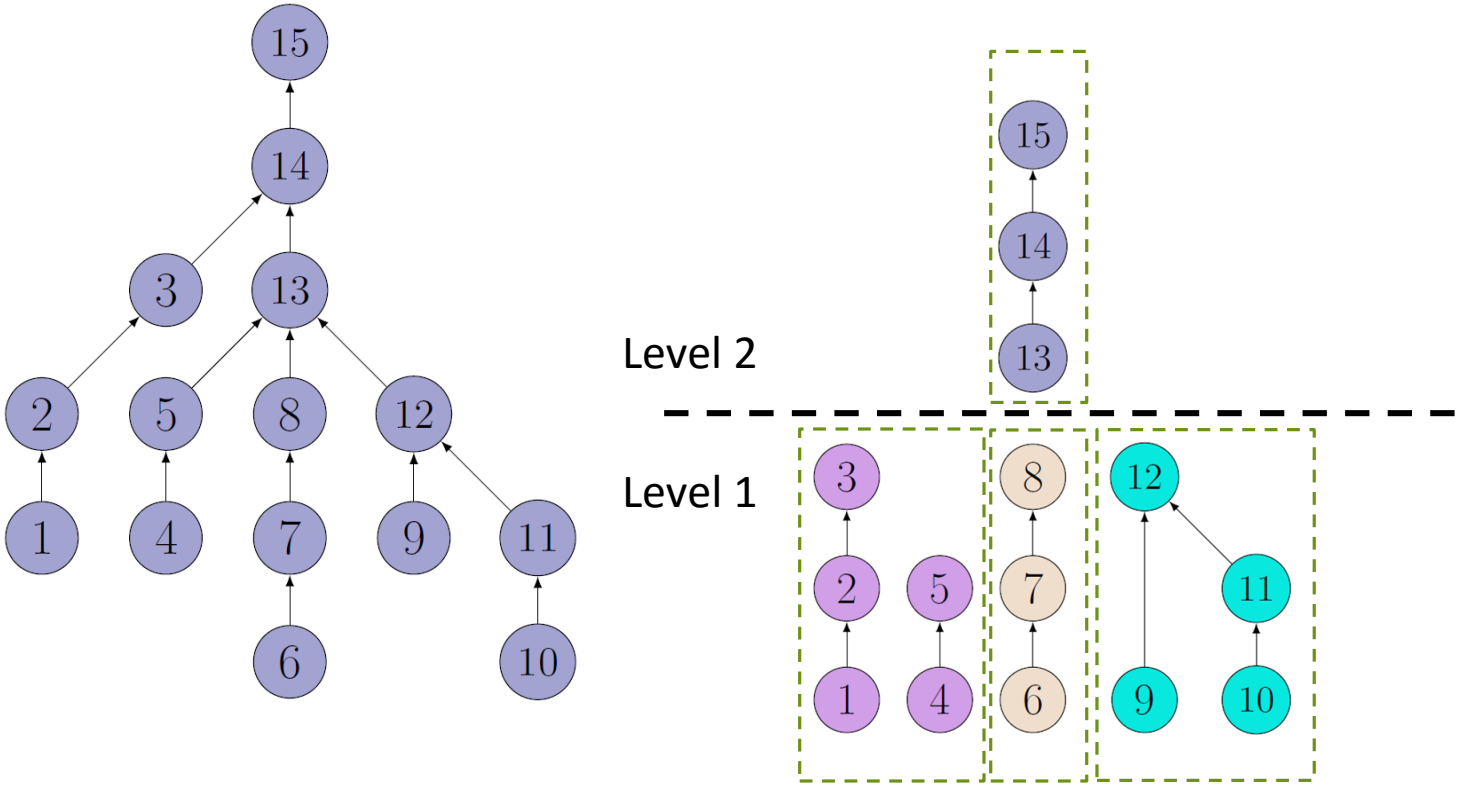


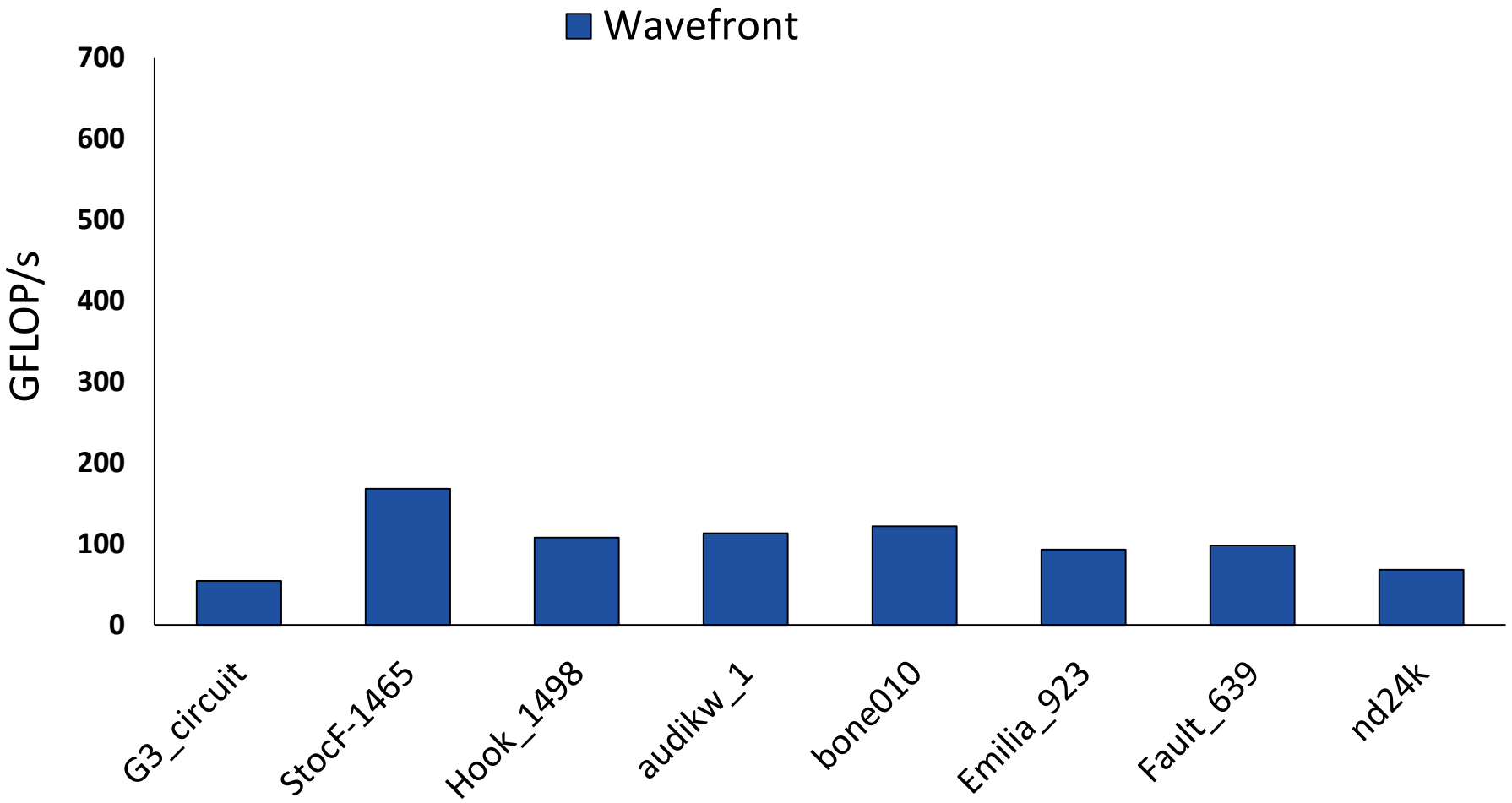

The level set created by Wavefront methods for the iteration space dependency graph of sparse Cholesky factorization.

# Loop Parallelism in ParSy

ParSy builds a coarsened level set to improve locality while providing a balanced enough parallelism.

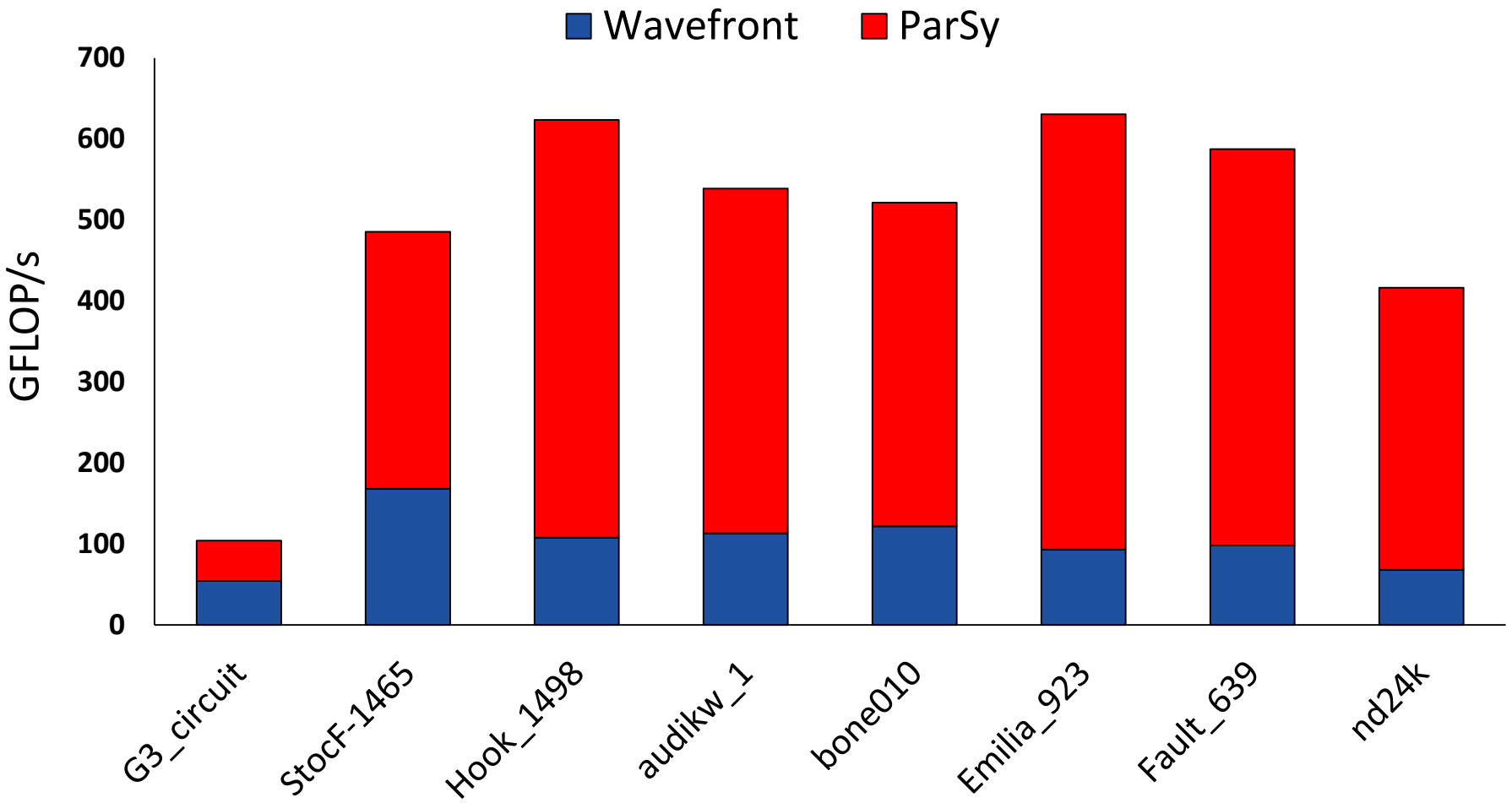


# CHOLESKY NUMERIC PERFORMANCE: WAVEFRONT



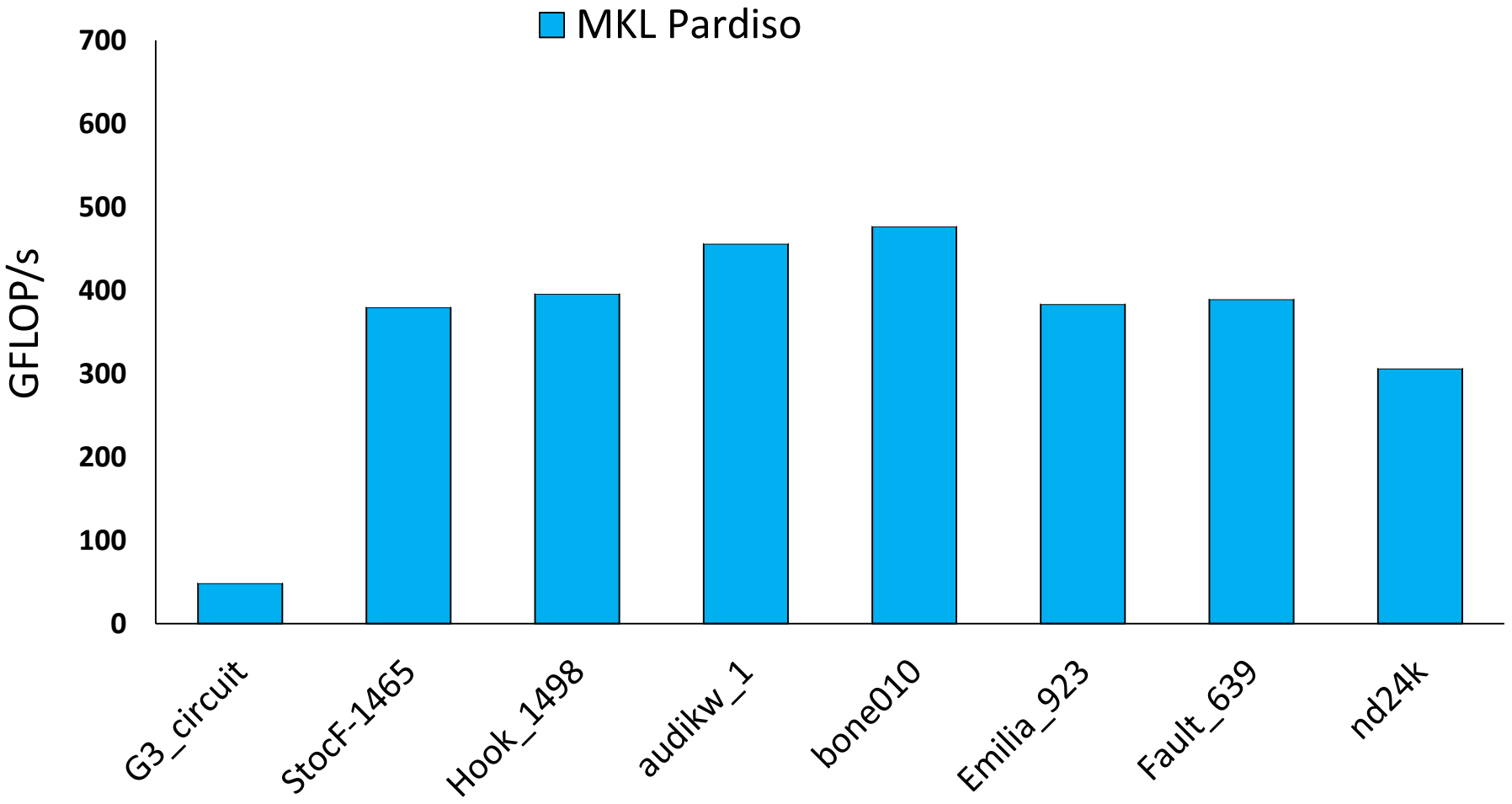

The blue bar shows the performance of Wavefront parallelism for selected benchmarks.

# CHOLESKY NUMERIC PERFORMANCE: WAVEFRONT



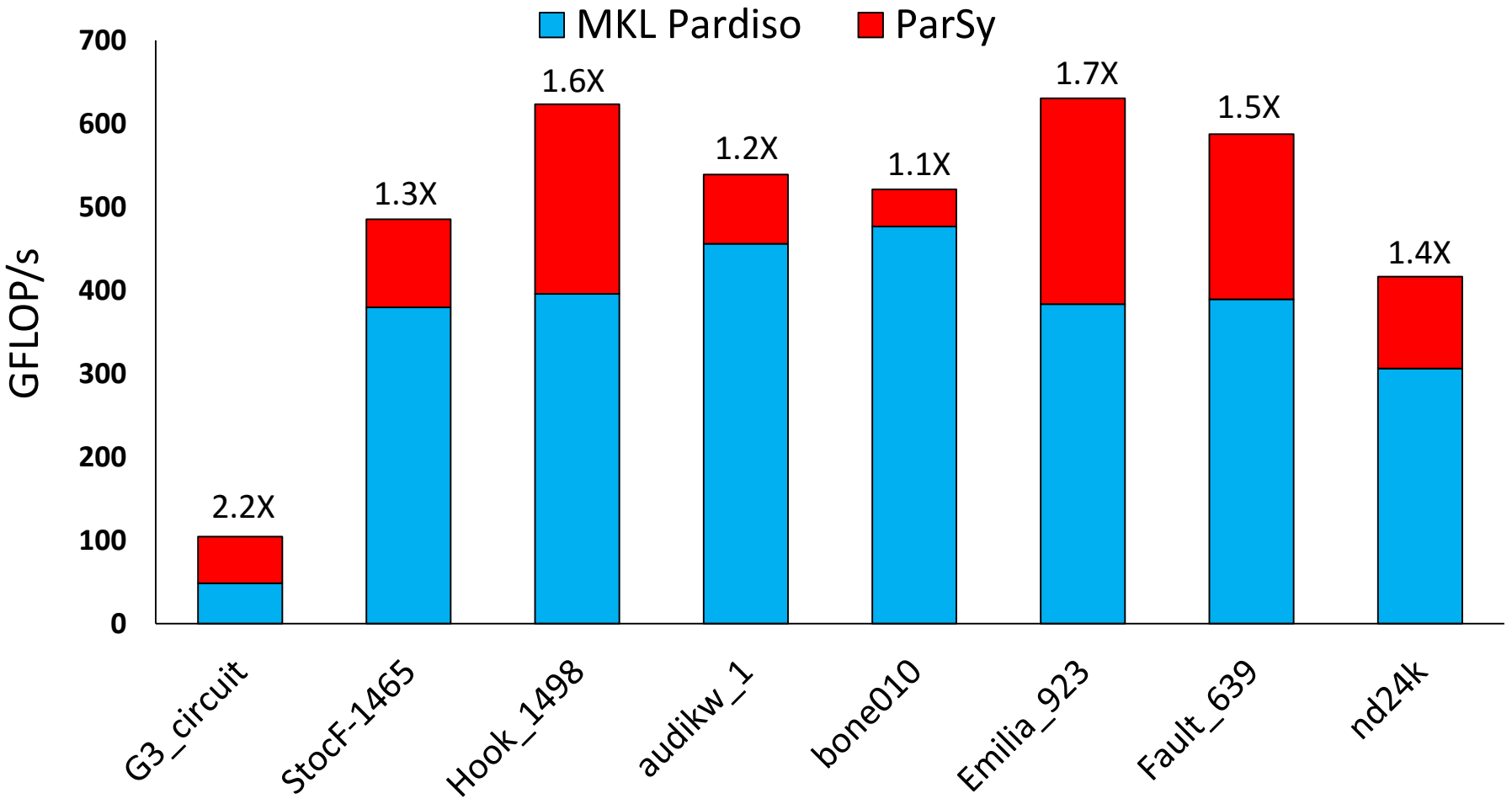

The red bar is the added performance from ParSy.

# CHOLESKY NUMERIC PERFORMANCE: MKL PARDISO




The blue bar shows the performance of MKL Pardiso for selected benchmarks.

# CHOLESKY NUMERIC PERFORMANCE: PARSY




The red bar is the added performance from ParSy.

# Outline

- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

# SYMPILER OVERVIEW

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

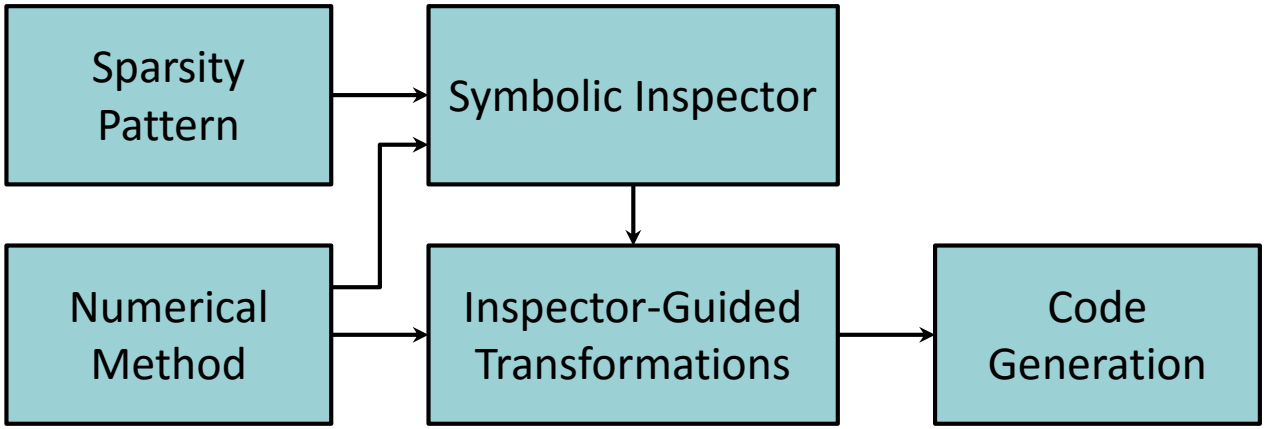
**Cheshmi, Kamil, Strout, Dehnavi "Sympiler: transforming sparse matrix codes by decoupling symbolic analysis." SC17*

# Implementation of ParSy

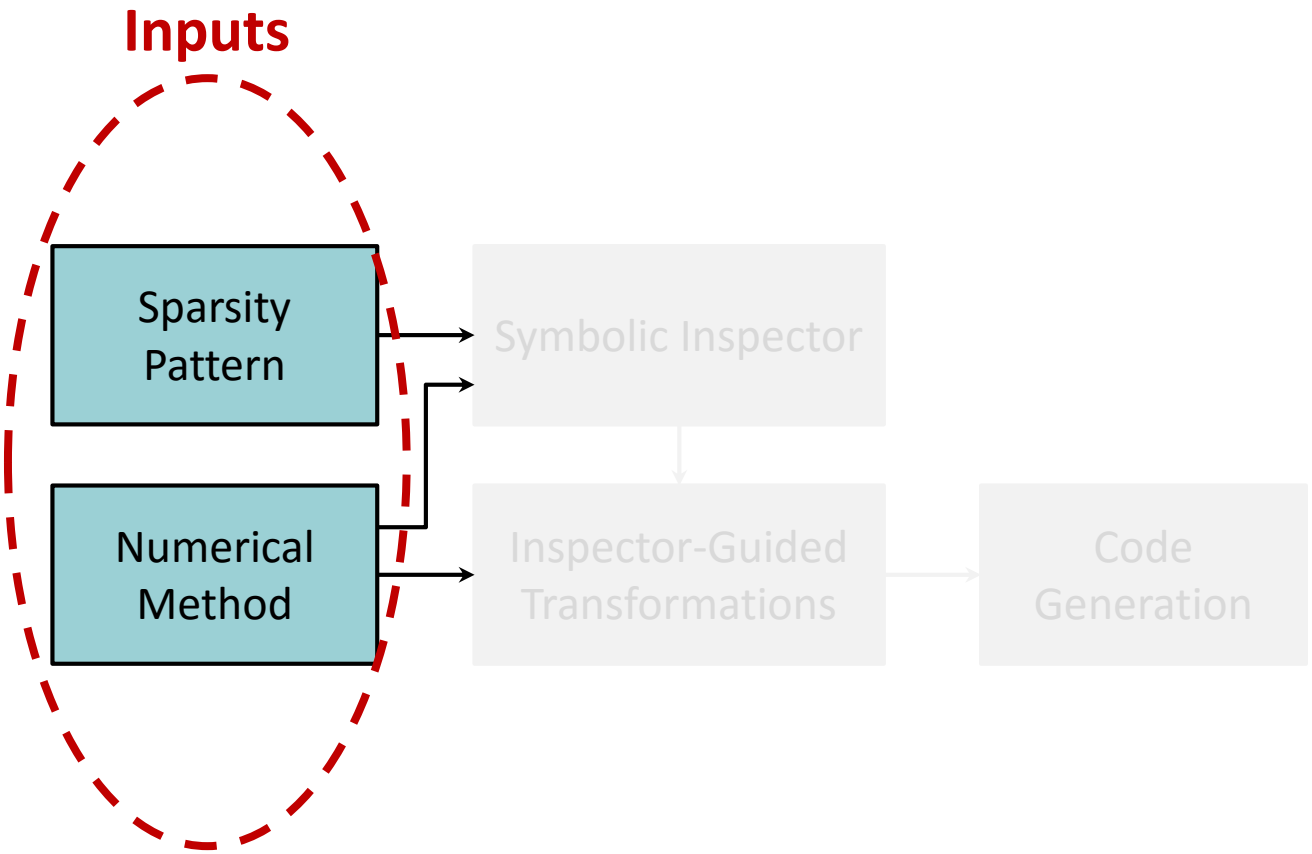
- **Sympiler*** is a domain-specific compiler for generating high-performance code for sparse solvers.
  - It uses symbolic information to transform the sparse code.
  - Sympiler does not support parallelism for multicore.
- ParSy generates parallel code for sparse matrix computations.
  - ParSy is built on top of Sympiler.
  - However, it can also be implemented at run-time.

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

# ParSy (and Sympiler) Internals

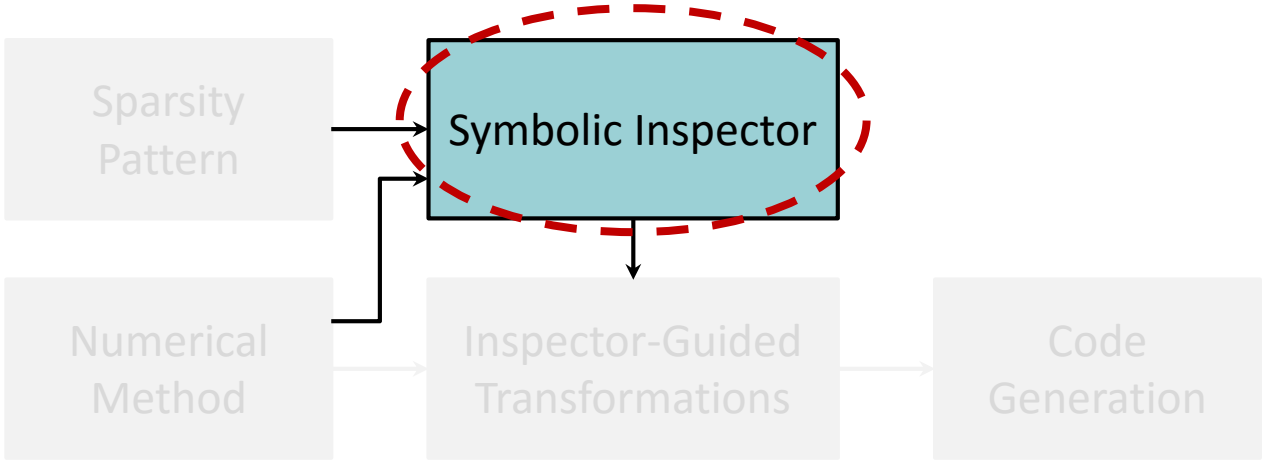


# ParSy (and Sympiler) Internals

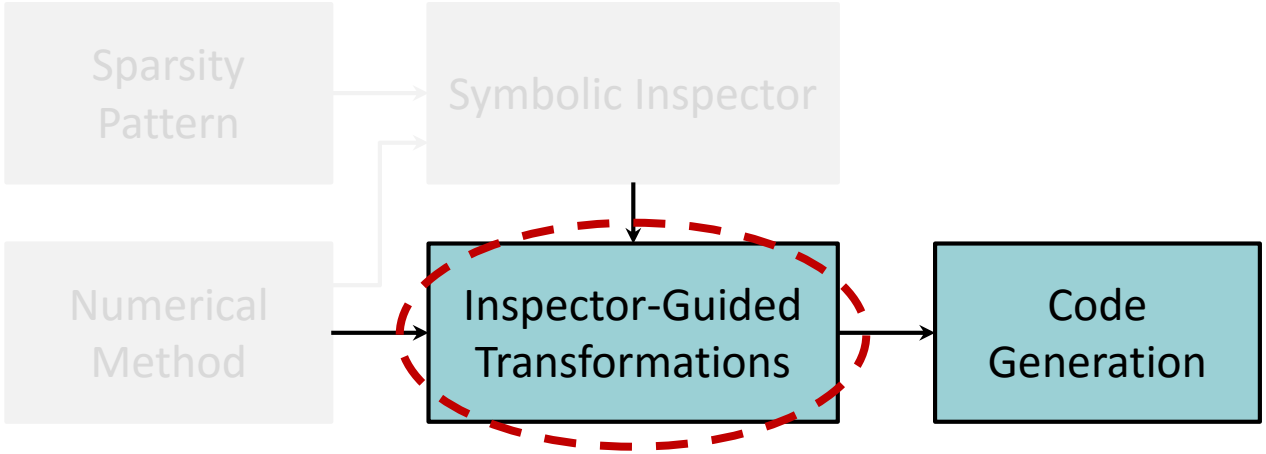


# ParSy (and Sympiler) Internals

**Inspection of the dependence graph to create *inspection sets***

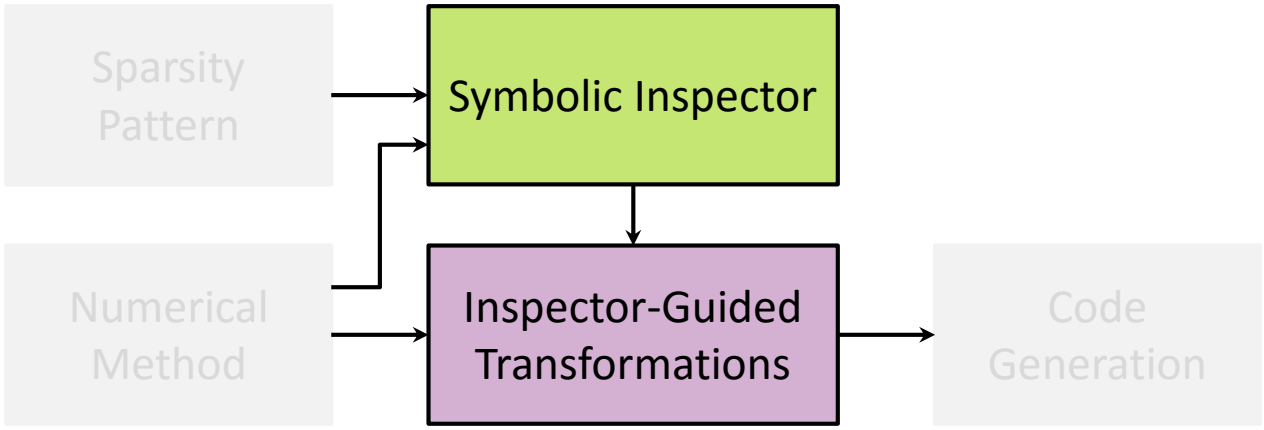


# ParSy (and Sympiler) Internals




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

# ParSy (and Sympiler) Internals




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

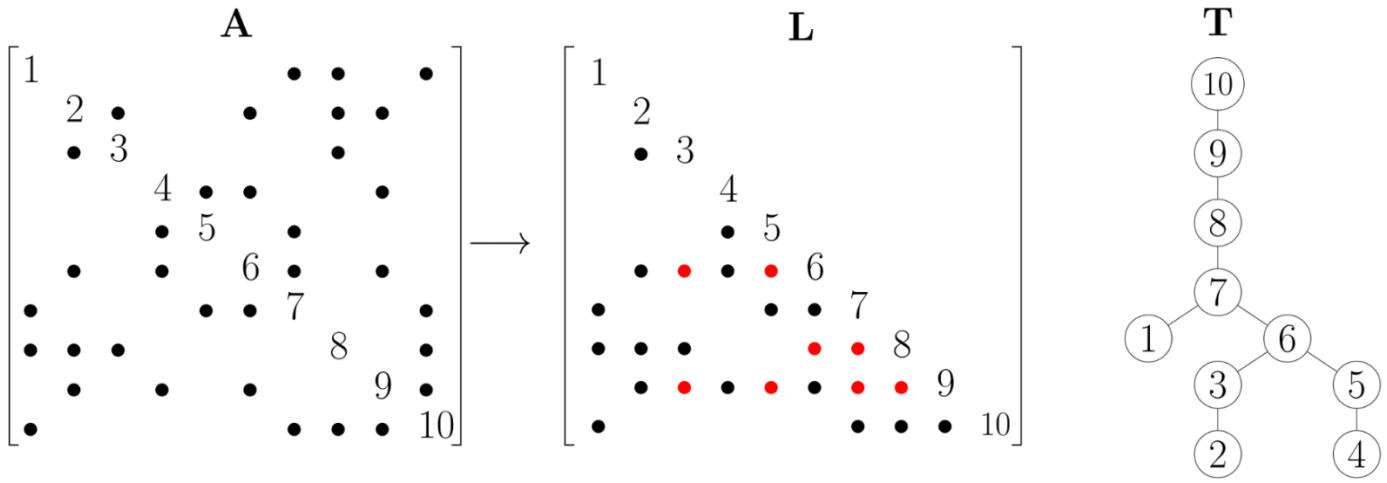
# OUTLINE

- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

# CHOLESKY FACTORIZATION

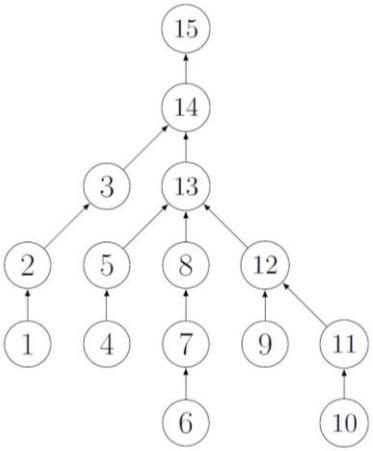
**Cholesky factorization** is commonly used in direct solvers and is used to precondition iterative solvers.

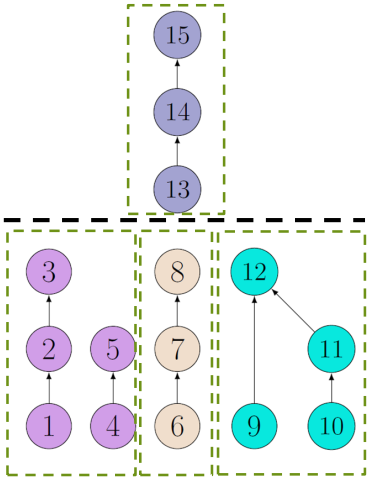
**The elimination tree (T)** is one of the most important graph structures used in the symbolic analysis of sparse factorization algorithms.



# 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**.



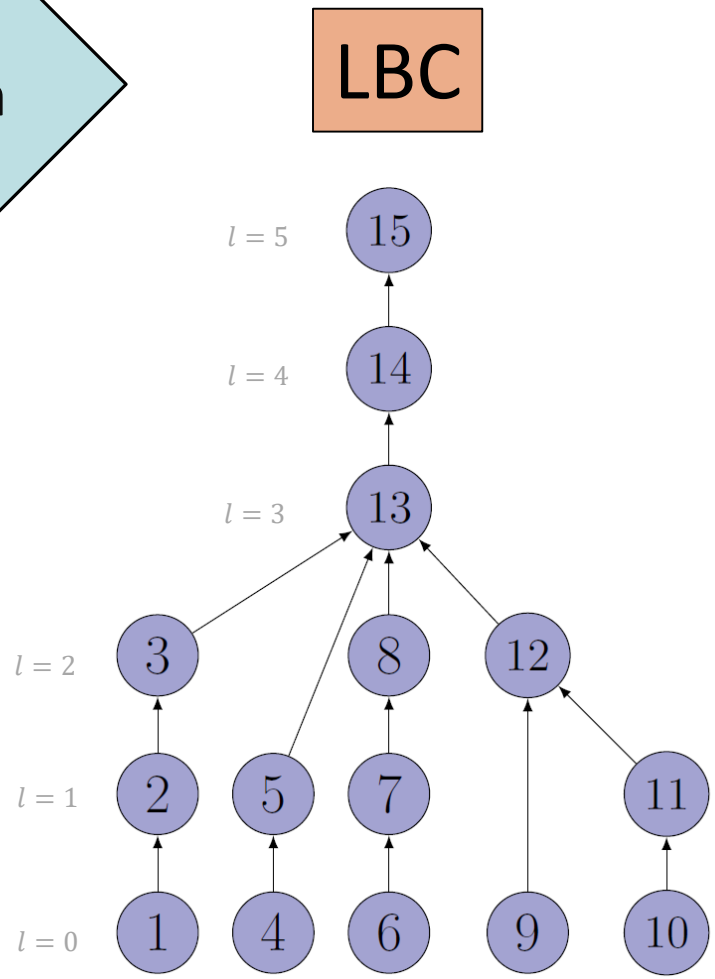

***H-Level Set*** = {{{ 1, 2, 3, 4, 5 }, { 6, 7, 8 } , { 10, 11, 9, 12 }} ; {{ 13, 14, 15 }} ; }

| **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



# The Load-Balanced Level Coarsening (LBC) Algorithm

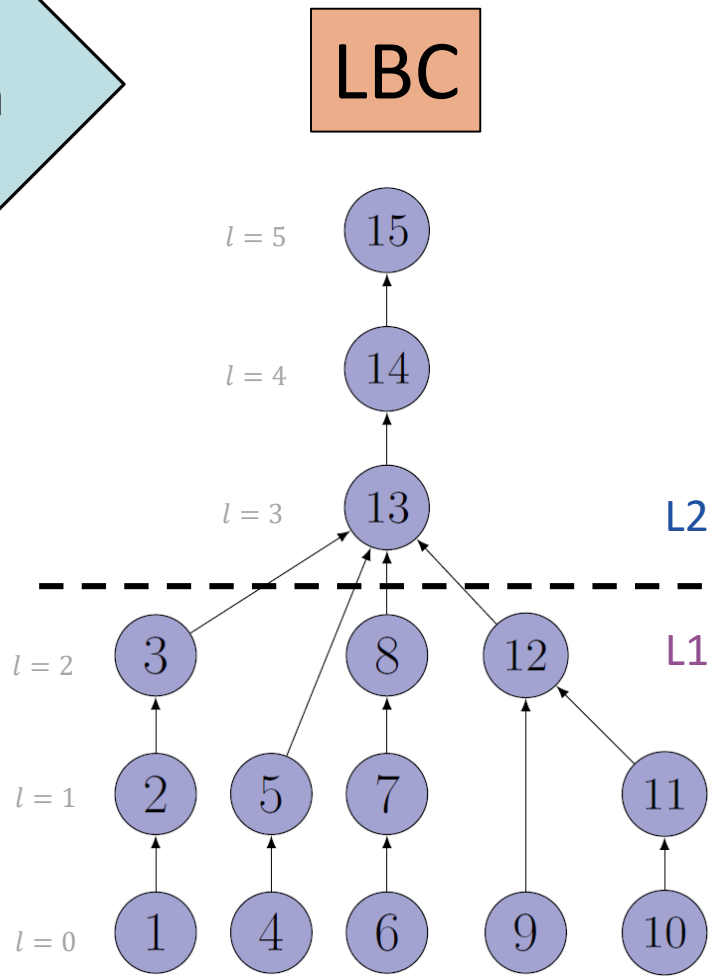
**Step 1:** L-partitioning

- Find the initial cut: first l-partition
- Build the rest of the l-partitions

$$l_{initCut} = \max(\{l | comp(G_{0:l}) \geq k\})$$

*Connected components*

*# of cores*



# The Load-Balanced Level Coarsening (LBC) Algorithm

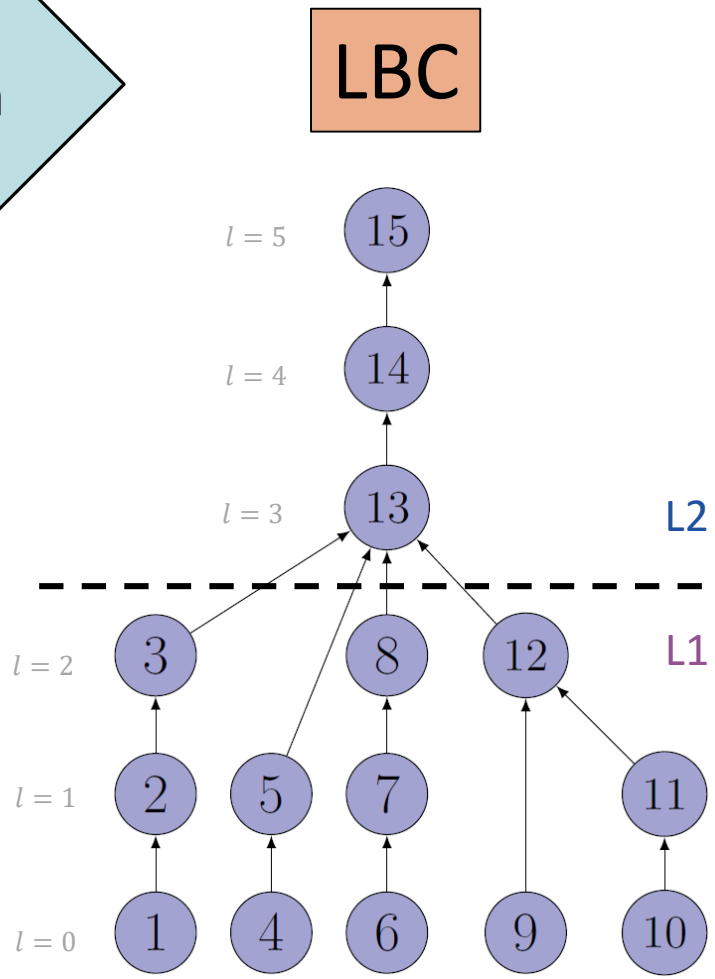
**Step 1:** L-partitioning

- Find the initial cut: first l-partition
- Build the rest of the l-partitions

$$l_{initCut} = \max(\{l | comp(G_{0:l}) \geq k\})$$

*Connected components* — *# of cores*

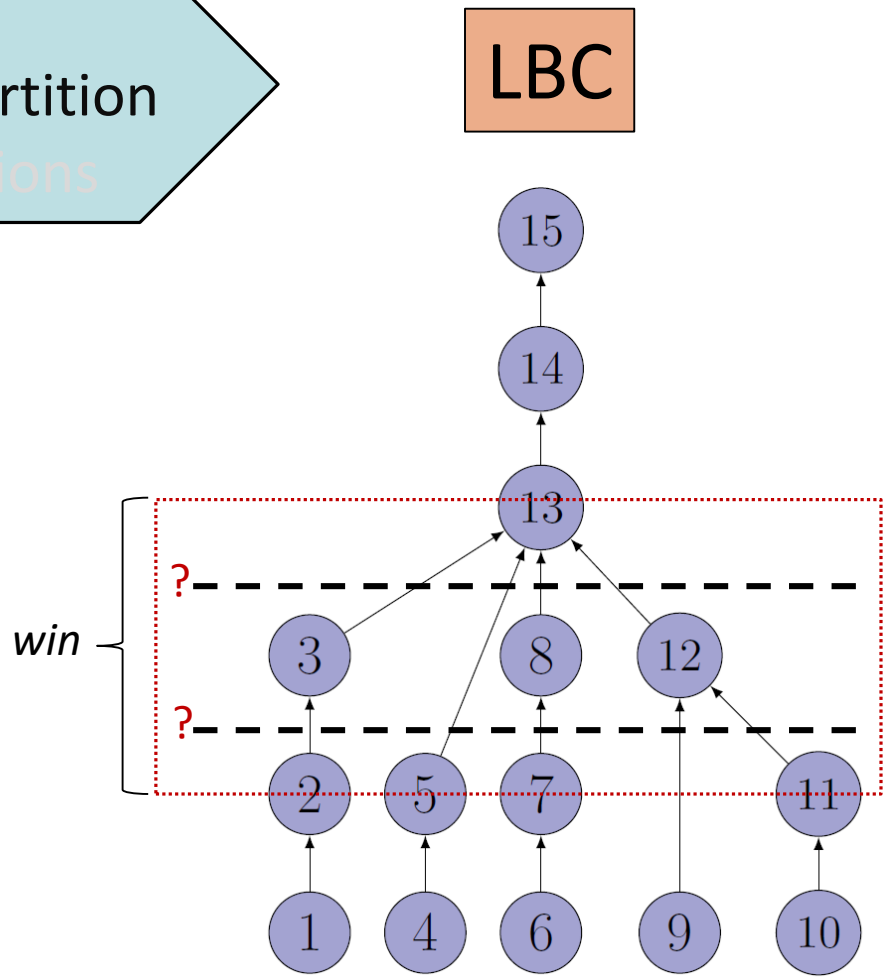
Creates good parallelism but what about load balance?



# 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



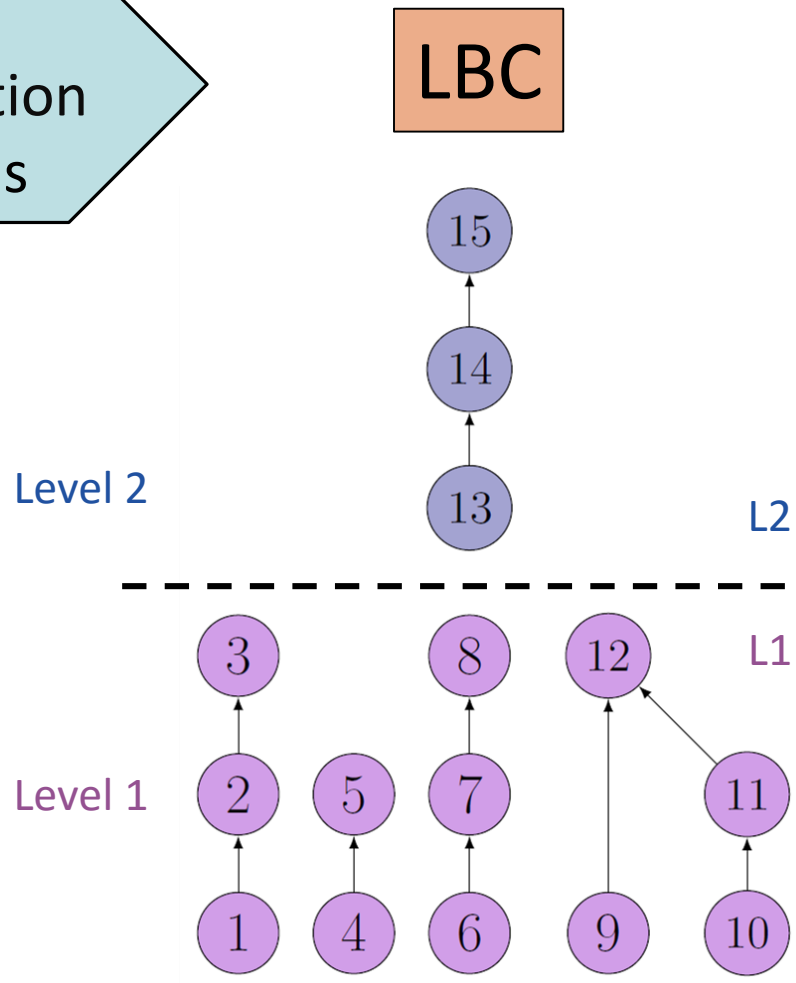

Using a *windowing heuristic* the initial cut is tuned to improve load balance.

A *proportional cost model* which uses the number of participating nonzeros in each node is used to measure load.

# 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



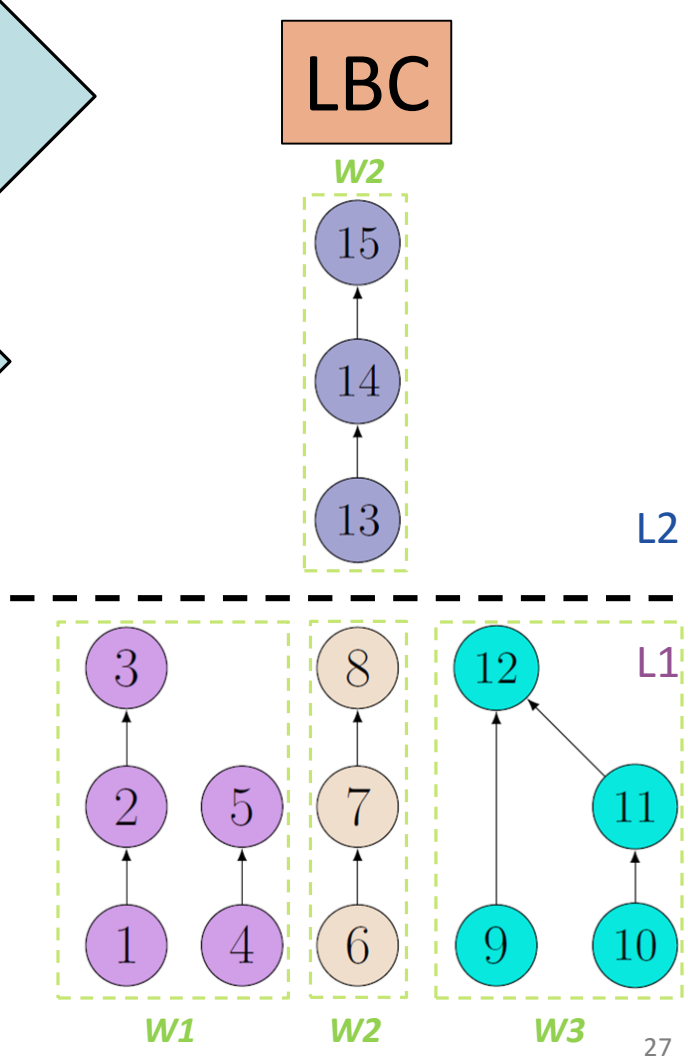
# The Load-Balanced Level Coarsening (LBC) Algorithm

**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

The *load balance constraint* ensures that the w-partitions within a level are balanced up to a threshold.

The *space-partition constraint* ensures that threads executing iterations in different w-partitions need not synchronize amongst each other.



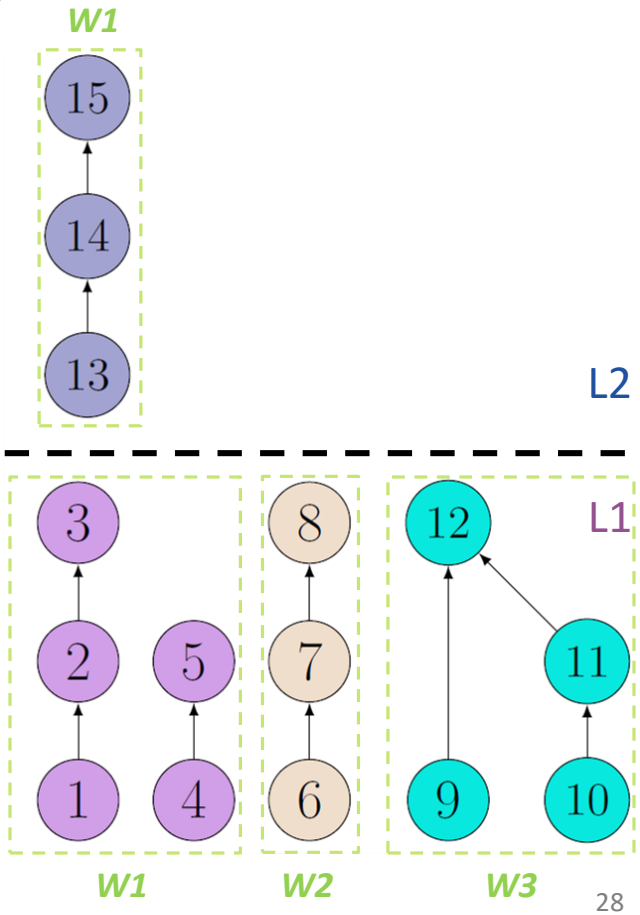
# The Load-Balanced Level Coarsening (LBC) Algorithm

**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




W-partitions with the same ID are assigned to the same core.

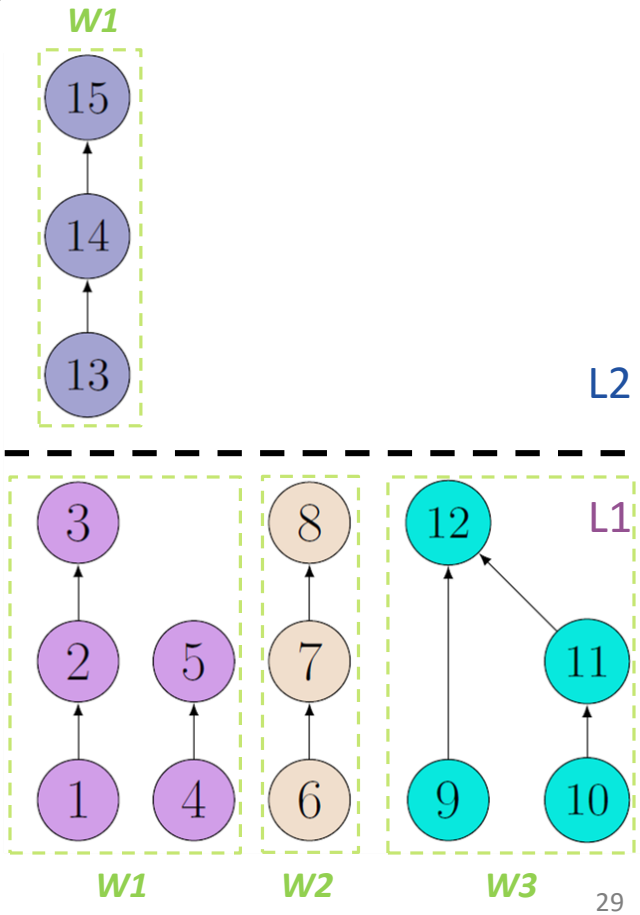
# The Load-Balanced Level Coarsening (LBC) Algorithm

**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 }} ; }

# Outline

- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

# 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)];
    }
}
```

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)];
            }
}}}
```

Internally annotated code

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

# Outline

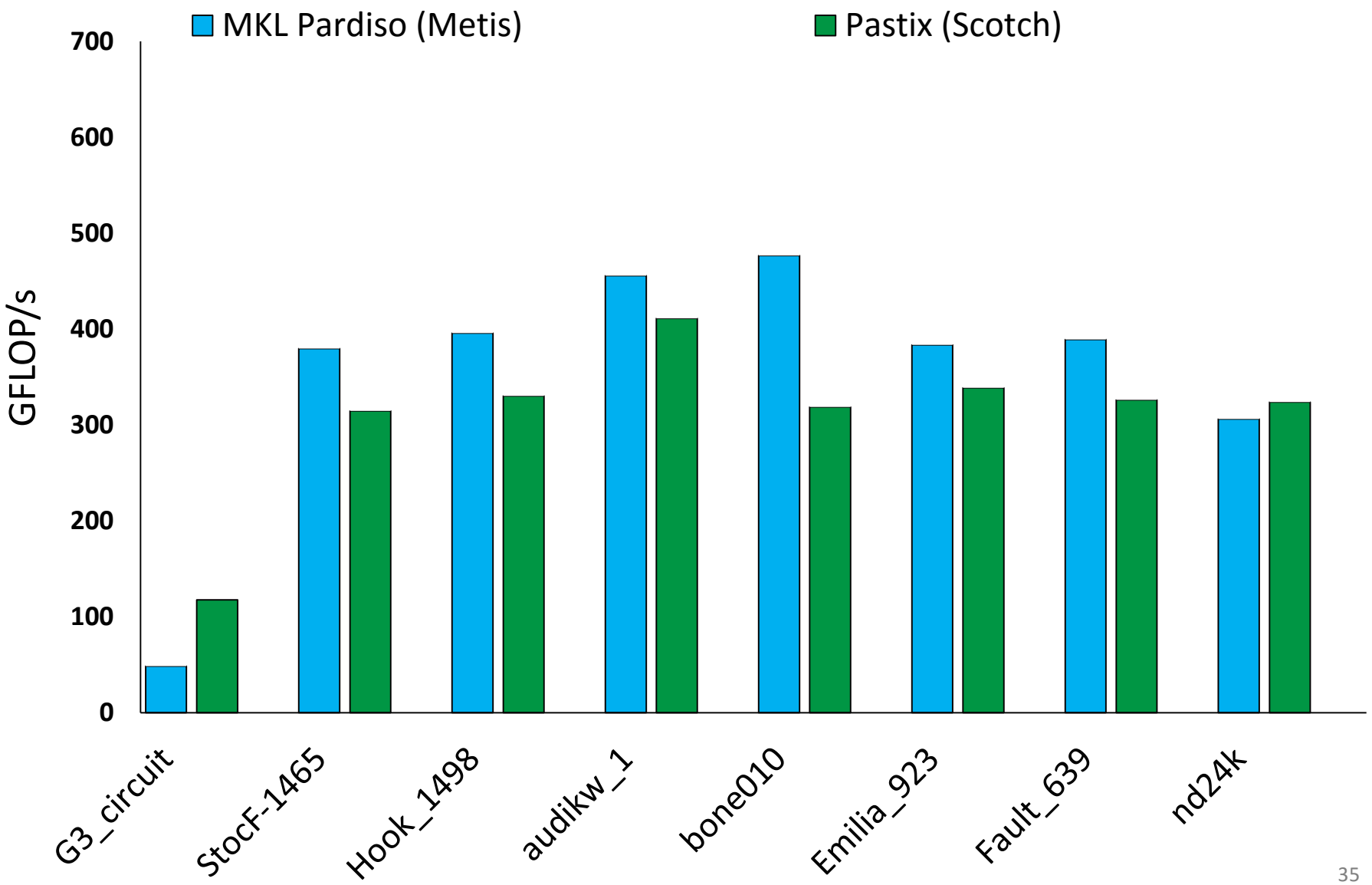
- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

# Experimental Setup

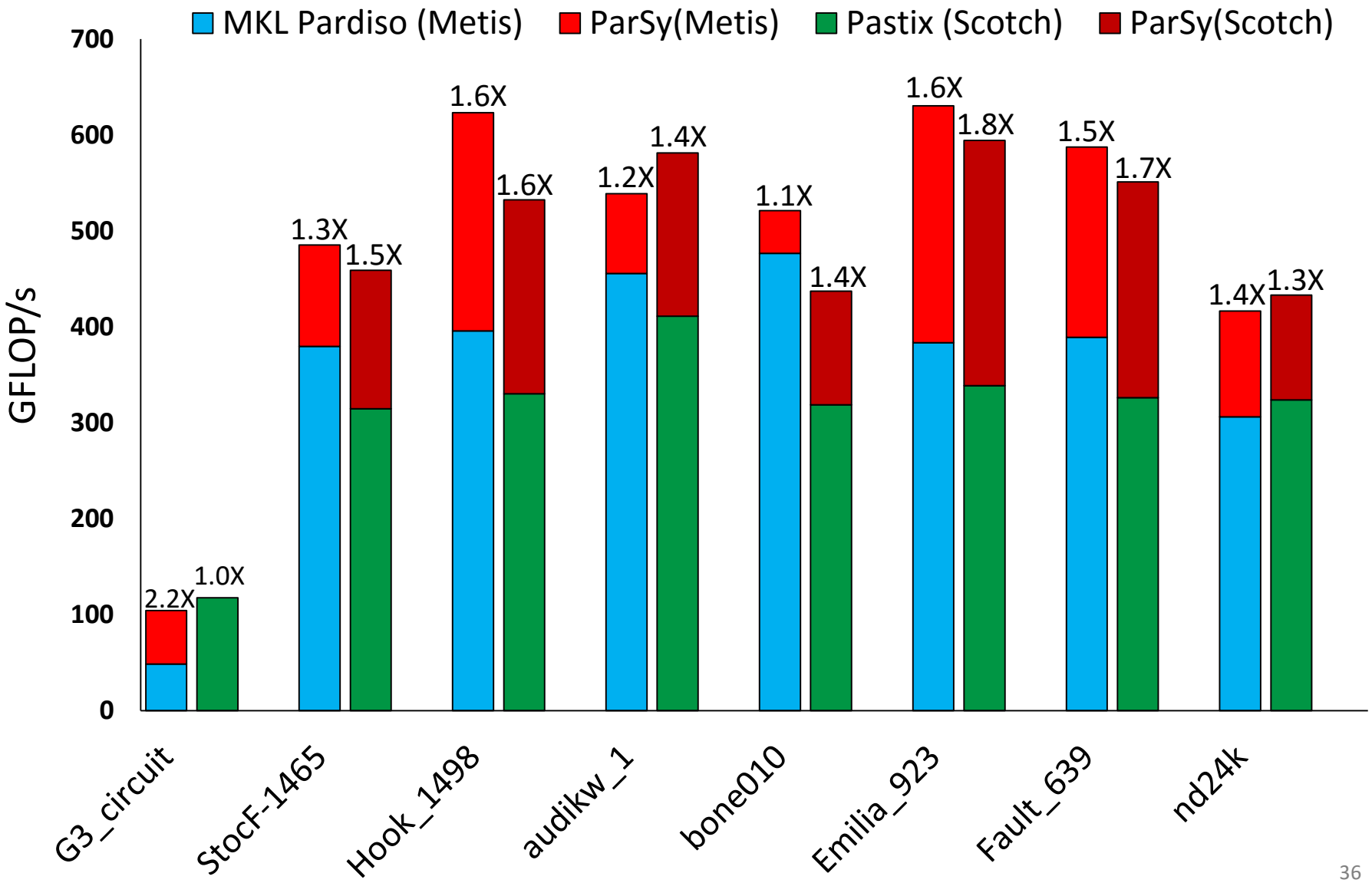
Numeric and symbolic times are compared separately where applicable; **Target processor**: Intel® Xeon® Platinum 8160 (Skylake); **Benchmarks**: Suitesparse matrix collection

| 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 |

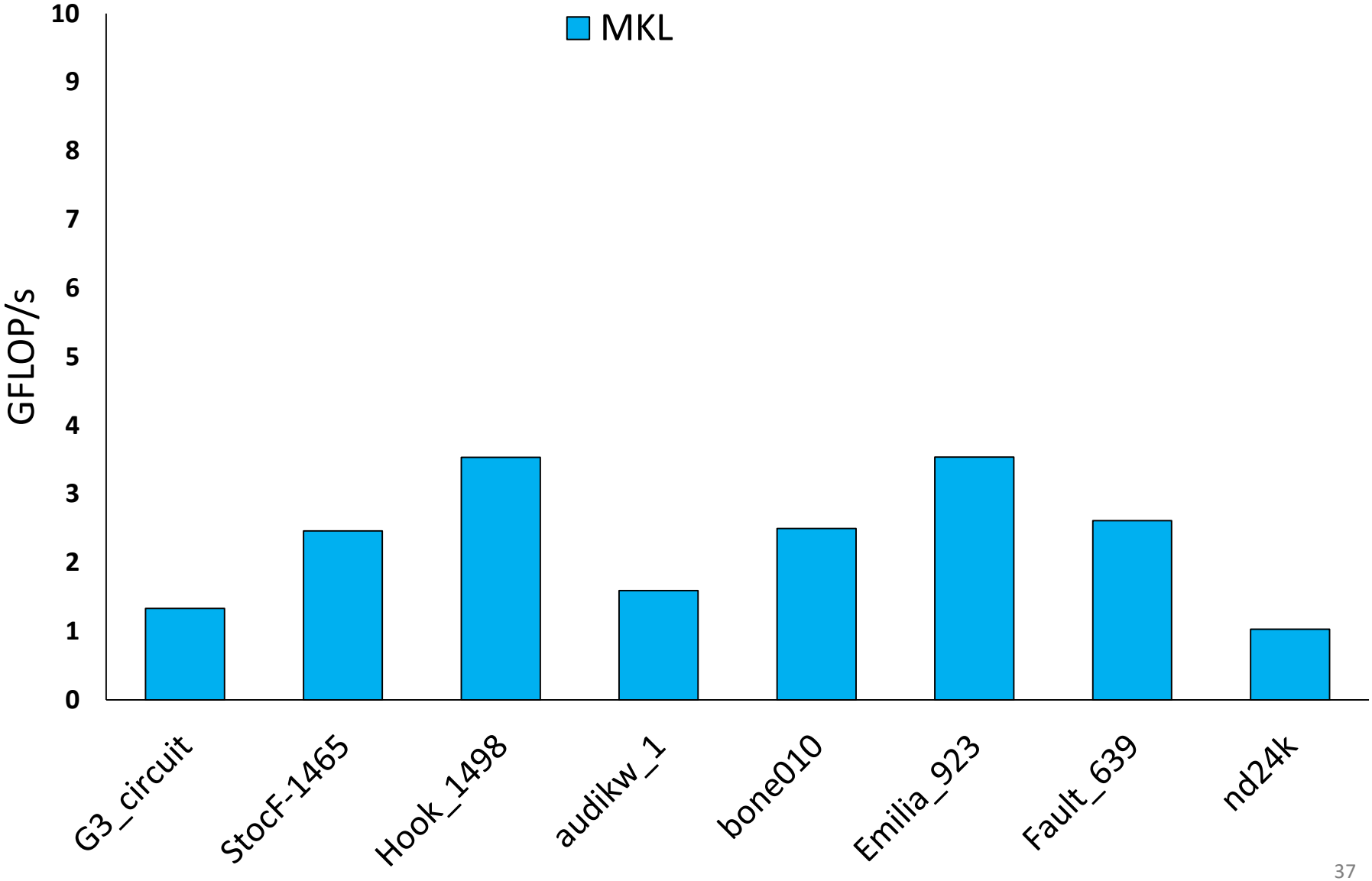
# PARSY VS LIBRARIES: CHOLESKY



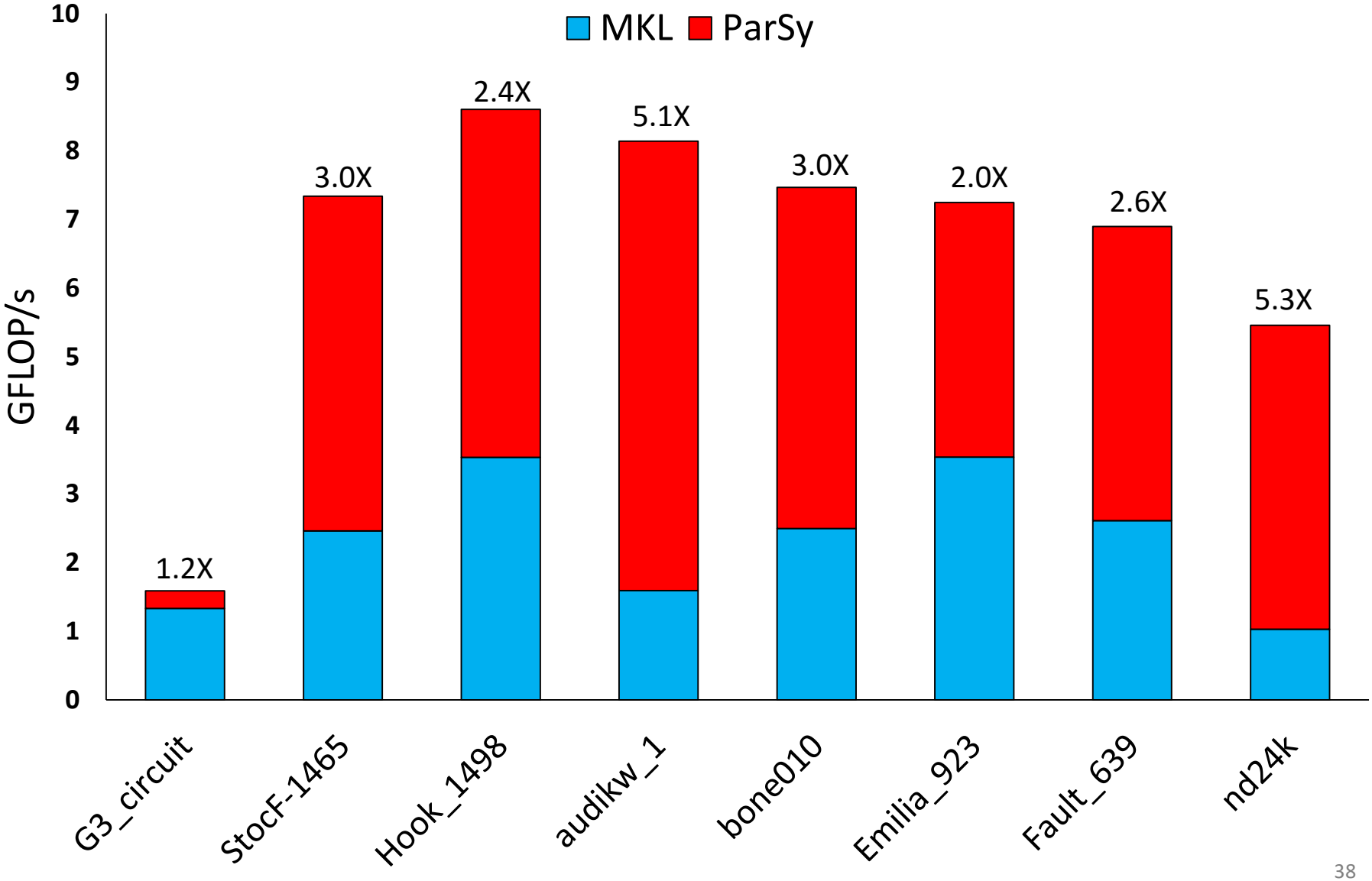
# PARSY VS LIBRARIES: CHOLESKY



# ParSy vs Libraries: Triangular Solve



# 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*

# Outline

- Overview
- ParSy: Inspection and transformation for parallelism
  - Sympiler internals and ParSy
  - H-Level inspection: The LBC algorithm
  - H-Level transformation
- Results
- Conclusion

# CONCLUSION

- ParSy is a domain-specific code generator for transforming sparse matrix codes for parallel multi-core processors.
- ParSy uses H-Level inspection and transformations to create coarse-level parallelism.
- The ParSy-generated code outperforms state-of-the-art sparse libraries Intel MKL Pardiso and PaStiX.
- ParSy's source code is publicly available from:

***https://www.Sympiler.com/***