Decoupling Symbolic Analysis from Numerical Factorization in Sparse Direct Solvers

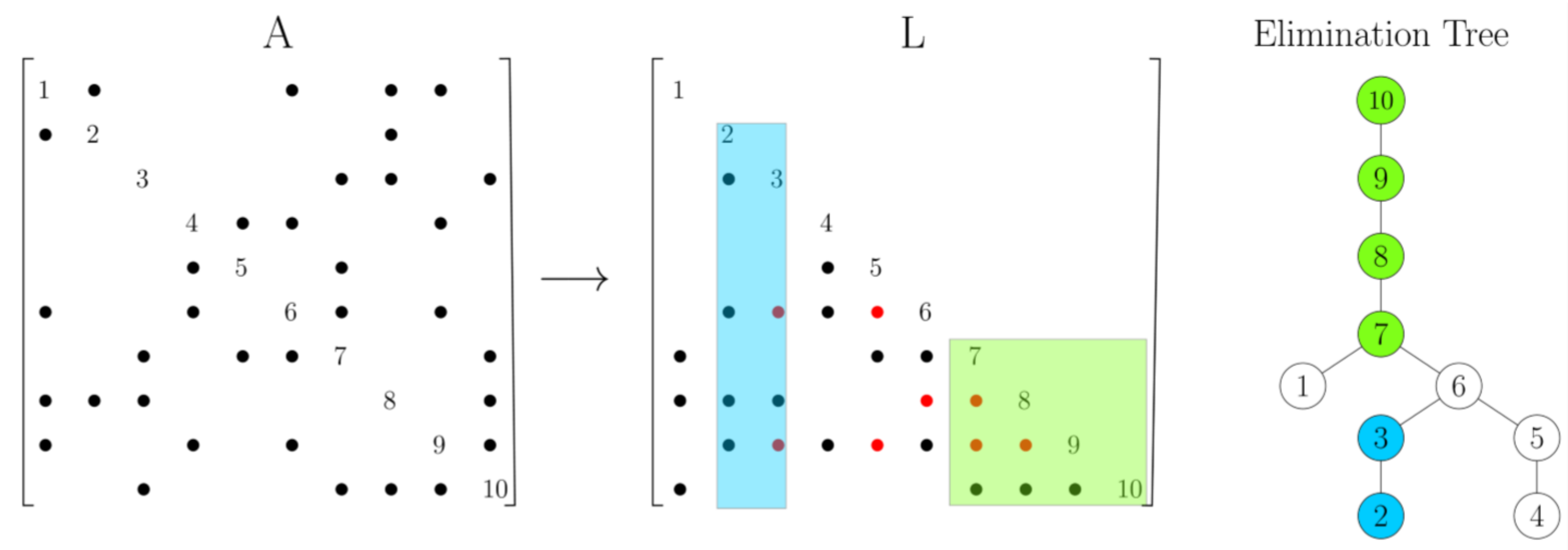
Kazem Cheshmi, Maryam Mehri Dehnavi
Department of Electrical and Computer Engineering, Rutgers University, New Brunswick NJ, 08854

Introduction

- Scientific simulations often need to find the solution to large sparse systems.
- Sparse matrix computations such as matrix factorization methods typically dominate the execution time of sparse solvers.
- **Sympiler,** decouples symbolic analysis from numerical computation at compile-time to enable the application of more aggressive code transformations.

Library	Compiler
❖Pros ✓Highly-optimized hand-tuned codes Cons ❑Hardware dependent ❑Difficult to maintain ❑Might not perform well for a different application	❖Pros ✓Easy to use for the domain expert ✓Cross-platform ❖Cons ❑Limited when transforming sparse codes ❑Available domain-specific compilers can only manipulate static index arrays

Symbolic Analysis in Direct Solvers

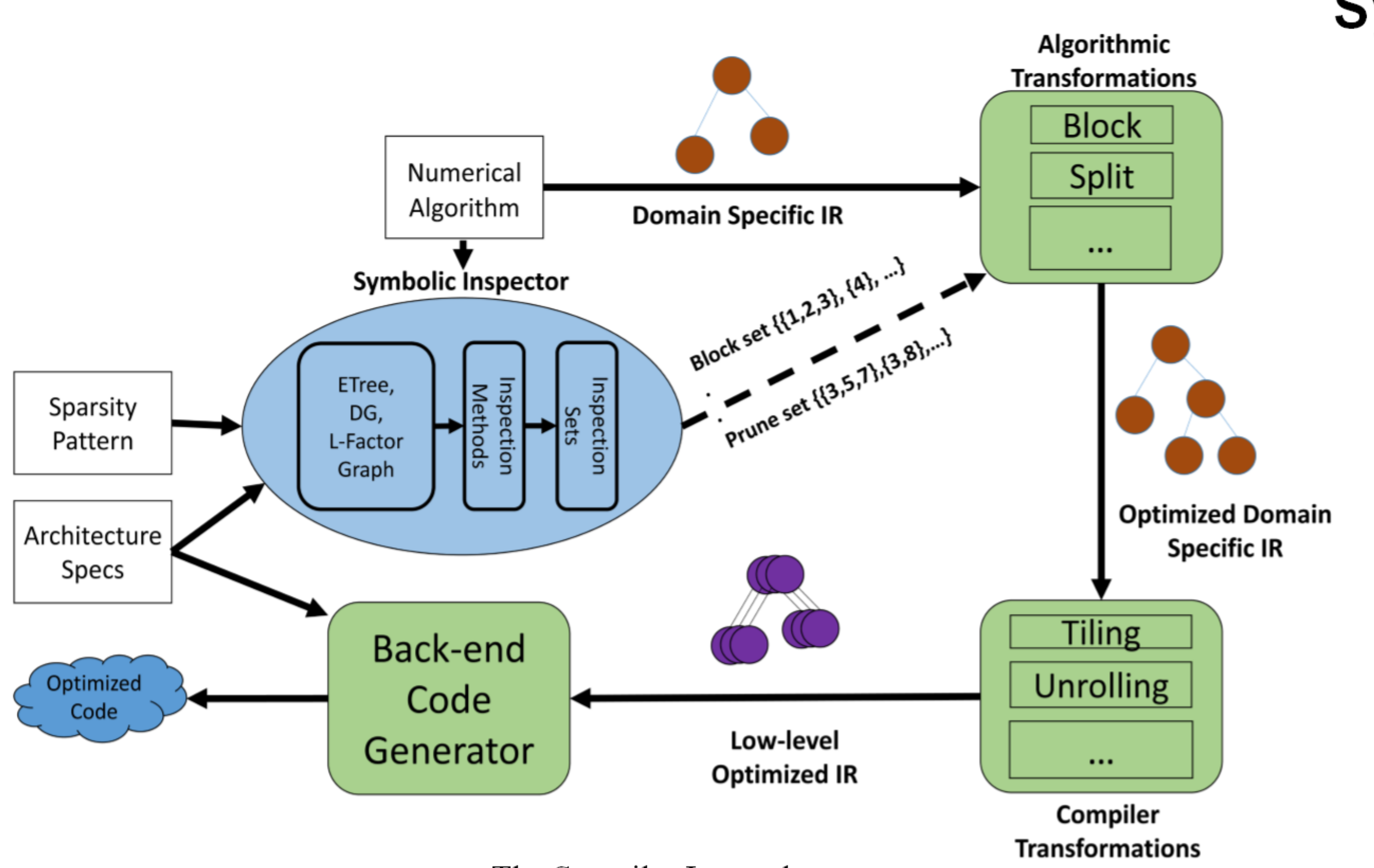


Cholesky Factorization:

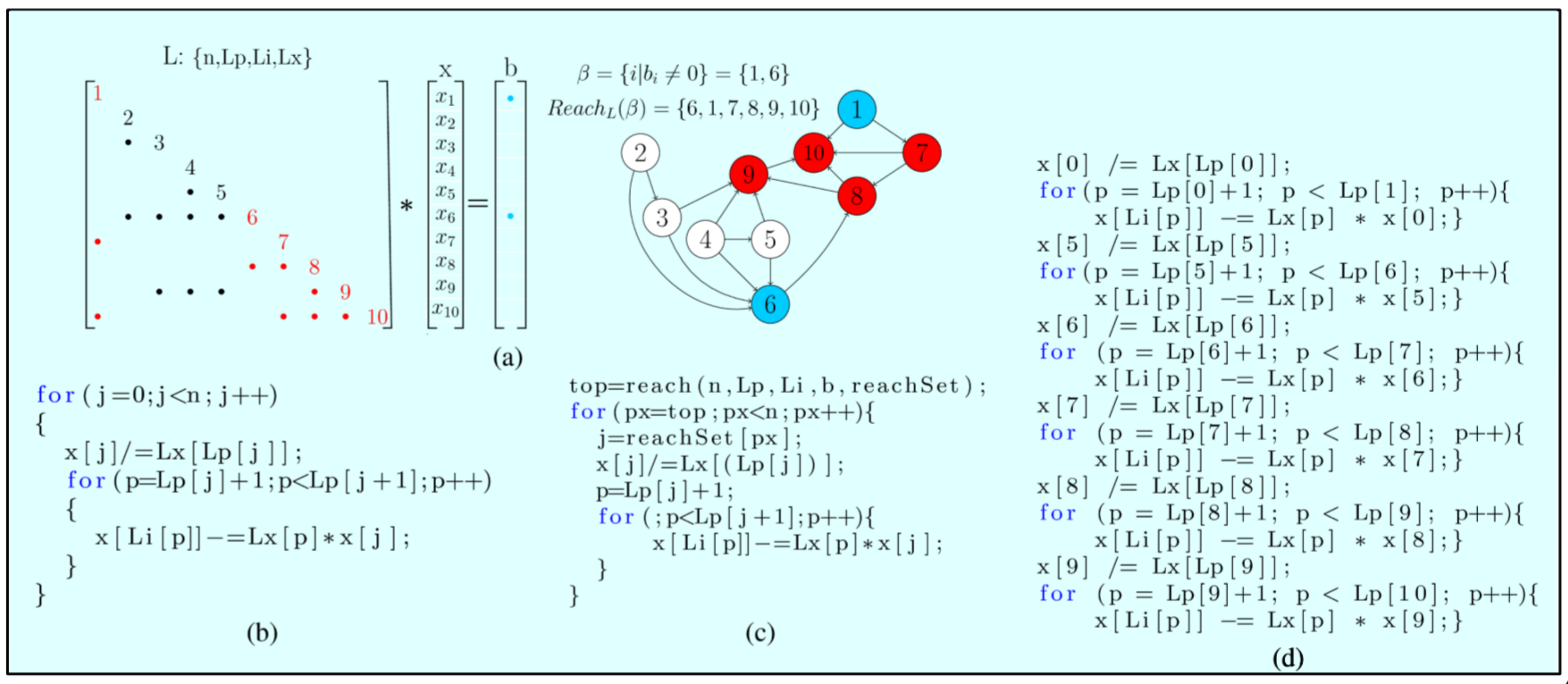
$$\text{Fill-in Computation} \longrightarrow L_j = A_j \bigcup \{j\} \bigcup \left(\bigcup_{j=parent(s)} L_s \setminus \{s\} \right)$$

$$\text{Supernode Detection} \longrightarrow |L_j| = |L_{j-1}| \wedge j = parent(j-1)$$

Sympiler Overview



The Sympiler Internal

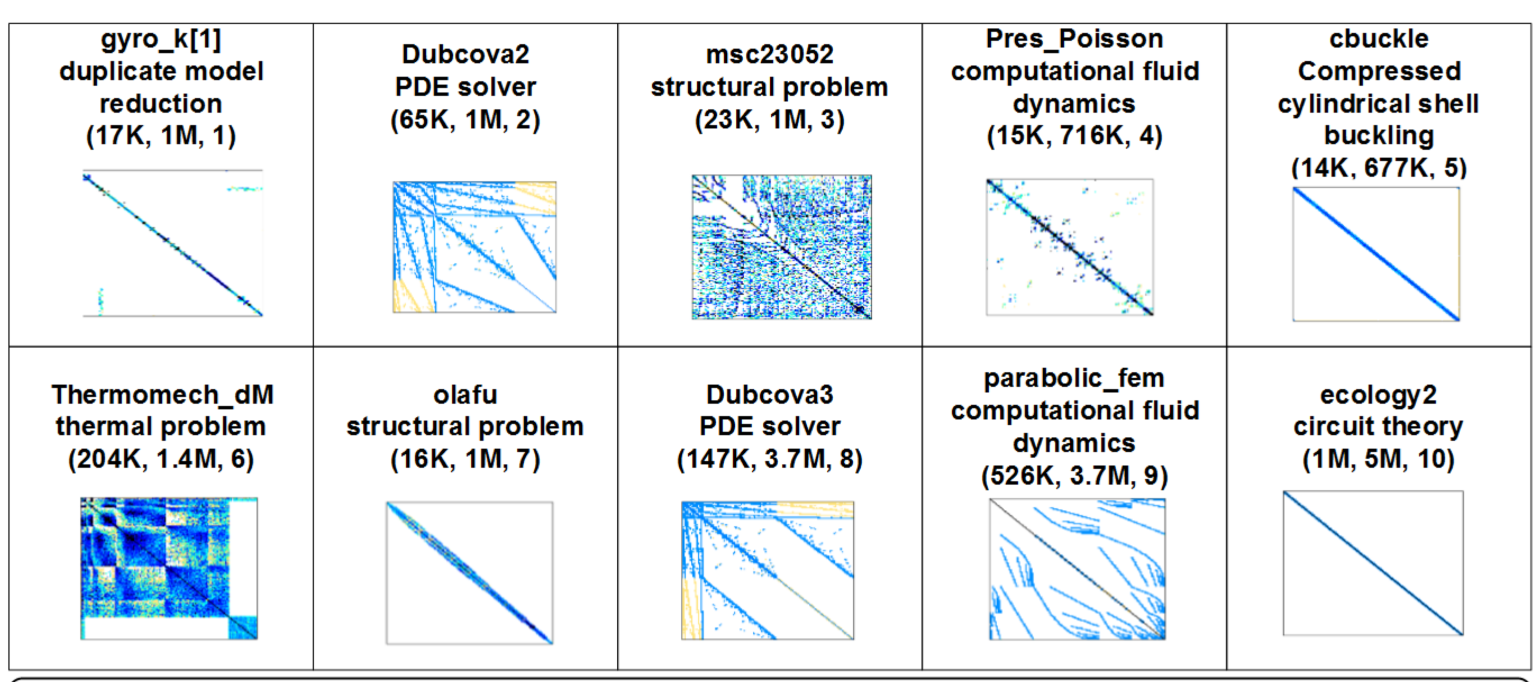


Motivating Scenario

Symbolic Inspector

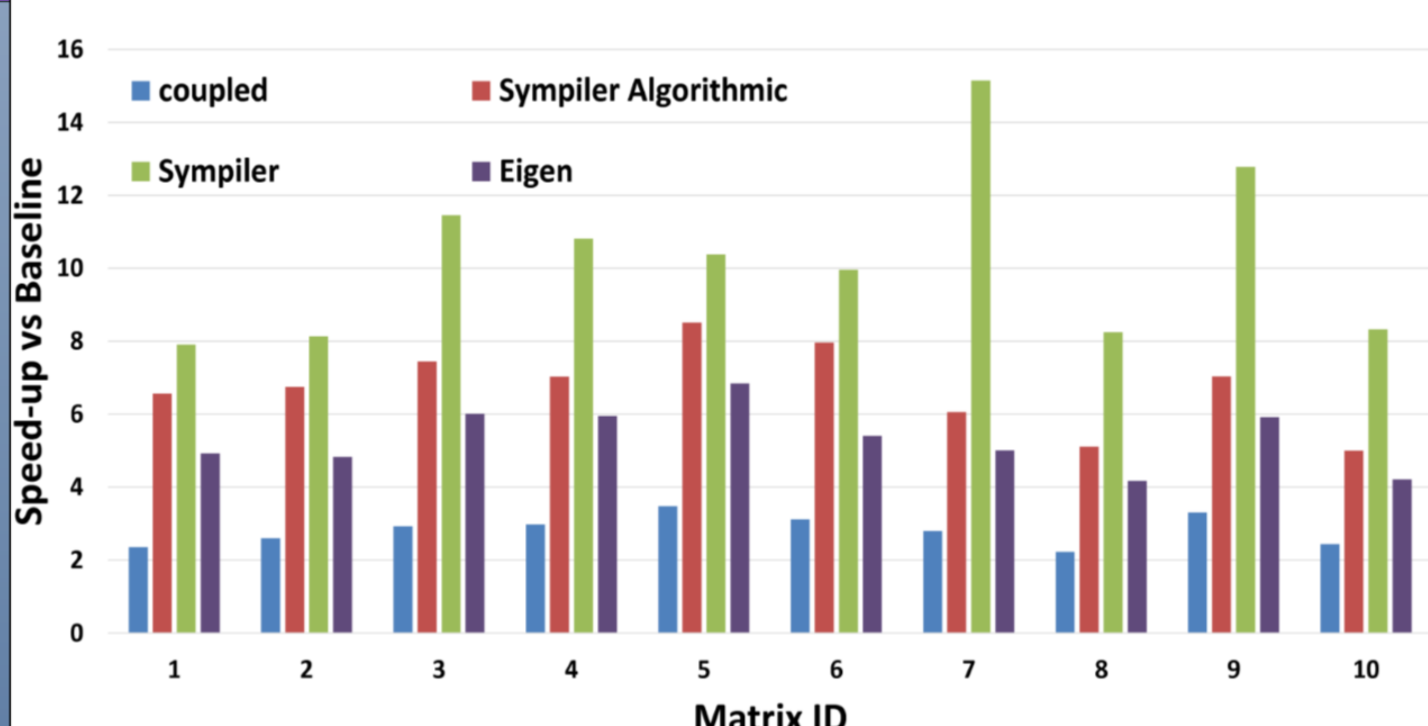
	Transformation	Inspection Graph	Inspection Method	Inspection Set
Triangular Solver	Variable iteration-space splitting	Dependence Graph + RHS Sparsity	DFS	Prune-set (reach-set)
Triangular Solver	2D variable size blocking	Dependence Graph	Node equivalence	Block-set
Cholesky	Variable iteration-space splitting	Etree + A Sparsity	Traversing up	Prune-set (reach-set)
Cholesky	2D variable size blocking	Etree + Col-Counts(A)	Comparing with parent	Block-set (Supernode)

Experimental Setup



Target Specification: Intel core i7-5820K, Clock: 3.30 GHz, Cache Size: 15 MB, Memory 32GB

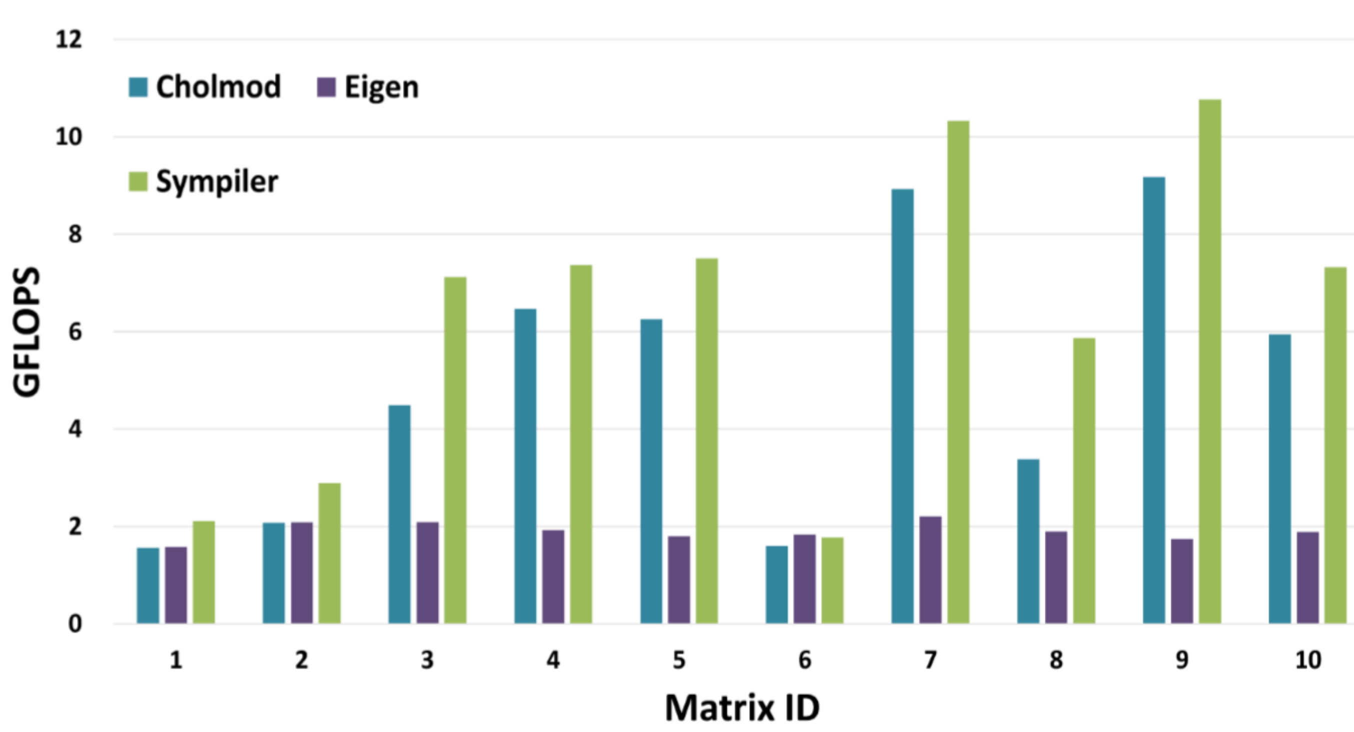
Results



The Effect of Split Transformation on Triangular Solver

Variable iteration-space splitting prunes the iteration-space and boost the performance for sparse right-hand-sides.

Enabled low-level transformation as a result of the split transformation is another source of performance gain.



Cholesky Factorization Performance Results

2D variable size blocking transformation breaks the sparse kernel into dense sub-kernels i.e., BLAS.

Sympiler outperforms specialized libraries such as CHOLMOD[2] and numerical libraries such as Eigen up to in order 1.74X and 6.1X.

Conclusion

❖Sympiler, decouples symbolic information from numerical manipulation at compile-time to enable the application of algorithmic- and low-level compiler transformations for sparse matrix methods. The Sympiler generated code outperforms state-of-the-art specialized libraries such as CHOLMOD up to 1.74X for Cholesky factorization.

❖In future work, we intend to extend the compiler to include more optimization techniques such as parallelization and vectorization. We also want to support more memory storage formats.

[1] T. A. Davis, and Y. Hu, "The university of Florida sparse matrix collection", ACM Transactions on Mathematical Software (to appear), http://www.cise.ufl.edu/research/sparse/matrices, January, 2009.

[2] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. ACM Transactions on Mathematical Software (TOMS), 35(3):22, 2008.