BERKELEY LAB

Parallelizing the Analysis Phase of Sparse Direct Linear Equation Solvers

Yen-Hsiang Chang

Qualifying Examination

Exam Committee: Katherine Yelick (Chair), James Demmel, Ming Gu, Xiaoye Sherry Li

Time & Location: May 5, 2026, 11am @ 373 Soda Hall

The Ubiquity of Sparse Linear Systems

Sparse linear systems drive progress in scientific computing, machine learning, data analytics, optimization, and many other domains.

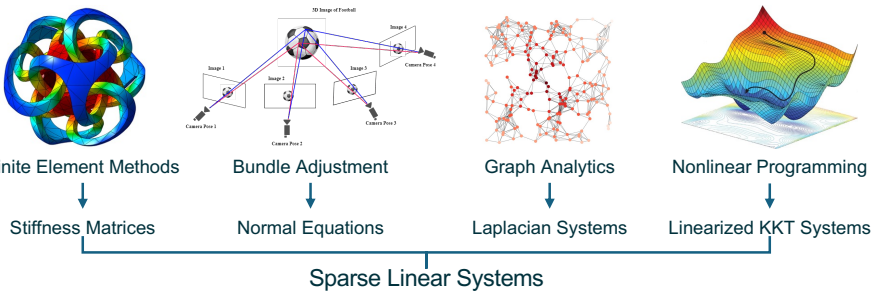


Figure 1: Applications relying on solving sparse linear systems. Images: [Glv17; Kum24; HSS08; ASKR18]

Two Paradigms: Direct vs. Iterative Solvers

Historically, solver selection forced a trade-off:

	Direct Solvers	Iterative Solvers
System Sizes	Small-to-Medium	Massive
Strengths	Robustness & Accuracy	Scalability & Memory Efficiency
Applications	Circuit Simulation [DP10] Nonlinear Programming [Duf04]	Graph Analytics [LB12] Reservoir Simulation [LVW00]

What if a system is both massive and ill-conditioned?

Direct Solvers at Scale

Fortunately, as technology and algorithms advance, direct solvers have become a viable option for solving large-scale systems, providing these advantages:

- **Black-Box Reliability:** Robustness without requiring domain-expert tuning.
- **Cost Amortization:** Factor once and reuse for multiple right-hand sides.
- **General-Purpose Preconditioning:** Achieved via incomplete factorization.

Scaling direct solvers is a thriving research direction, benefiting both paradigms to handle the massive, ill-conditioned systems that modern science demands.

The Anatomy of Direct Solvers

Consider solving a sparse linear system $Ax = b$ where A is sparse. A typical direct solver consists of four steps:

1. **Reordering** that reduces fill-in or extracts special structure.
2. **Symbolic factorization** that determines the nonzero structures of the factors.
3. **Numerical factorization** that computes the factors.
4. **Triangular solves** that perform forward and backward substitution.

Why focus on the “analysis phase” (reordering & symbolic factorization)?

Motivation 1: Less Research in the Analysis Phase

SPARSE DIRECT SOLVER RESEARCH IN THE LAST DECADE

STAGE 1: REORDERING

Application Specific:
- Mesh: [ZMIM25, ZMD+26]

Refinement:
- Reduce Off-Diagonal Blocks: [PFRR17]

Machine Learning:
- Algorithm Selection: [TJC+25]
- Quality-Performance Tradeoff: [GHS+22, DK23, LNY+26]

STAGE 2: SYMBOLIC PHASE

GPU(s):
- Cholesky: [RA23]
- LU: [GLL21, XJAR23]

Portability:
- Utilizing GraphBLAS: [SLB25]

STAGE 3: NUMERICAL PHASE

Solver Updates:
- SuperLU_DIST: [SLVL15, SLV18, SLV19, LLLS23]
- PaStiX: [PDF+18]
- MUMPS: [PG25]

New Solver & Technique:
- GLU: [PT20]
- STRUMPACK/ButterflyPACK: [LXG+21, LGCL20, AGB+22, GS22]
- symPACK: [BJN+23]
- PanguLU: [FZW+23]
- Caracal [RZH+25]
- Trojan Horse: [LZN+26]
- cuDSS [Nv26]

Application Specific:
- Batched Band Linear Systems: [NAA25]
- Optimal Power Flow: [ŚKR+24]
- Conic Optimization: [CTN+24]

STAGE 4: TRIANGULAR SOLVES

Single GPU:
- Scheduling Design: [LZ20, LNL20, FDE24, GHGF26]

Distributed-Memory:
- Communication Design: [LJGL18, SKLV19, DLWL21]

Machine Learning:
- Algorithm Selection: [DEFQ21, HSLS24]

Figure 2: Research in sparse direct solver in the last decade.

Motivation 2: The Analysis Phase Is Now a Bottleneck

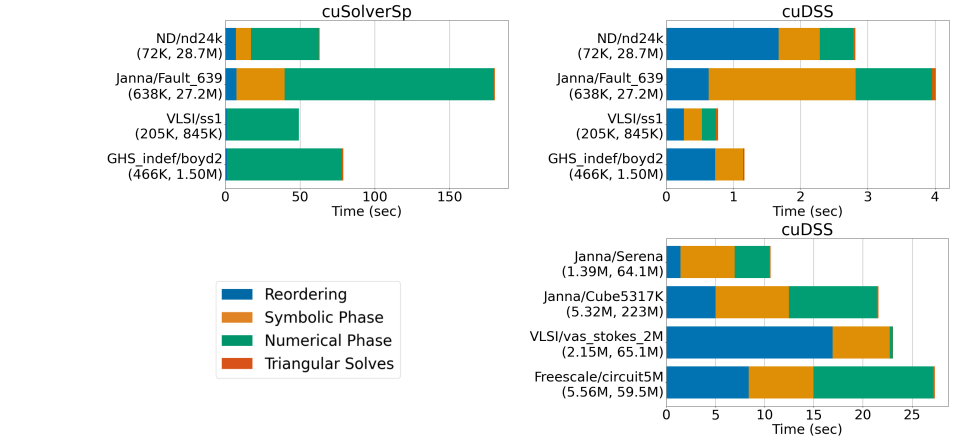


Figure 3: Performance breakdown of cuSolverSp (v11.7.4.40) and cuDSS (v0.7.1) solving linear systems in double precision using default settings on an A100-80GB GPU, where cuDSS is the successor of cuSolverSp.

Presentation Agenda

These motivations lead to the proposed thesis theme:

Parallelizing the Analysis Phase of Sparse Direct Linear Equation Solvers

In this talk, we will:

- **Explore** the design space of parallelizing the analysis phase.
- **Present** our previous work on parallelizing the approximate minimum degree ordering algorithm.
- **Propose** research directions for the thesis.

Design Space Exploration

What Does Reordering Do?

Matrix reordering (rows and/or columns) achieves:

- **Numerical Stability** by matching and/or pivoting.
- **Sparsity Preservation** by reducing fill-in (nonzeros in the factors).
- **Structure Extraction** by revealing low-rank off-diagonal blocks.
- **Performance Optimization** by reorganizing the execution plan.

How Does Reordering Interact with Different Kinds of Sparse Direct Solvers?

Table 1: Interactions between sparse direct solvers and the reordering purposes.

	LL^T (SPD)	LDL^T (Symmetric Indefinite)	LU (Unsymmetric)
Numerical Stability	-	1×1 or 2×2 Pivots	MC64/Pivoting
Sparsity Preservation	ND/AMD/COLAMD		
Structure Extraction	BLR/HSS		
Performance Optimization	Post-Ordering and Tree-Balancing		

We will primarily focus on preserving sparsity.

Two Heuristics to Preserve Sparsity

Preserving sparsity in the factors is often treated as a graph problem. Two heuristics are commonly used:

- **Nested dissection** uses a divide-and-conquer strategy to identify small vertex separators that partition the graph into independent subproblems.
- **Minimum degree**-based algorithms use a greedy heuristic that selects the vertex with the fewest current neighbors for elimination at each step.

These heuristics mainly work for symmetric matrices. For unsymmetric matrices, heuristics are applied to $|A^T||A|$ or $|A| + |A^T|$.

Obstacles of Parallelization for Reordering

Shared- [LK15] and distributed-memory [KK98; CP08] nested dissection tools exist, but they remain difficult to port to GPUs due to their deeply recursive nature and significant control-flow divergence.

- There are GPU k-way partitioners [GMBR24; LLJ+26], but an efficient GPU nested dissection method has yet to be implemented.

There are no efficient parallel minimum degree-based algorithms in the literature as the greedy heuristic is inherently sequential.

- We will present our work [**C**BD26] on this shortly.

Design Space of the Reordering Phase

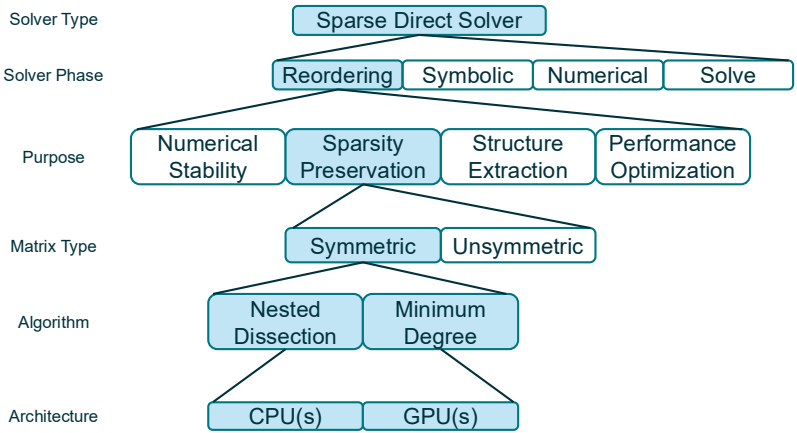


Figure 4: One branch of the design space of parallel reordering methods for sparse direct solvers.

What Does Symbolic Factorization Do?

After the input matrix is reordered, symbolic factorization computes the nonzero structures of the factors and allocates suitable data structures.

Unlike reordering, symbolic factorization is not universal for two primary reasons:

- **Algorithm-Specific Outputs:** The data structures generated during symbolic factorization are typically tailored to the specific numerical algorithm.
- **Numerical Interdependency:** The symbolic and numerical phases may be interleaved to accommodate dynamic pivoting for numerical stability.

Symbolic Factorization Algorithms

There are mainly two types of algorithms to compute the nonzero structures:

Table 2: Two types of algorithms for symbolic factorization.

	Implicit Algorithms	**Explicit Algorithms**
Data Requirement	Original matrix	Previous factors
Memory Allocation	In one go	On the fly
Characteristics	Complexity is comparable, but incremental algorithms tend to be faster [GL93].	
Examples	[RT78; Sch82]	[GP88; GL93]

Recent Papers on Parallel Symbolic Factorization

Recent papers target GPU acceleration for the symbolic phase.

- They prefer implicit algorithms since there is more parallelism and less dependency.

Paper	Matrix	Algorithm	Architecture
[GLL21]	Unsym	Implicit	GPUs
[RA23]	Sym	Implicit	CPU or GPU
[XJAR23]	Unsym	Implicit	GPU
[SLB25]	Unsym	Implicit or Explicit	CPU or GPUs

Issues in Parallelizing Symbolic Factorization

Recent GPU solutions still rely on CPU:

- While the final factors may fit within GPU memory, the intermediate structures required by the parallel algorithms often exceed the available capacity.
  - The amount of memory needed scales with the amount of parallelism.
- Some parts of the algorithm still stay on CPU.

None of the recent implementations have been integrated into existing solvers, nor have they been fairly compared against one another.

Design Space of the Symbolic Phase

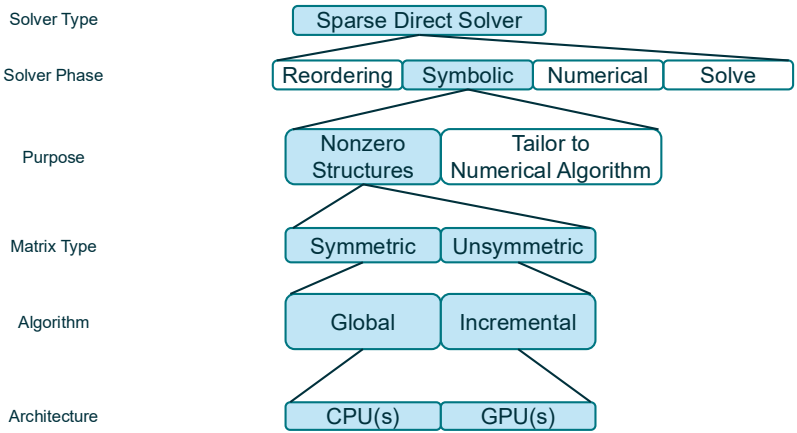
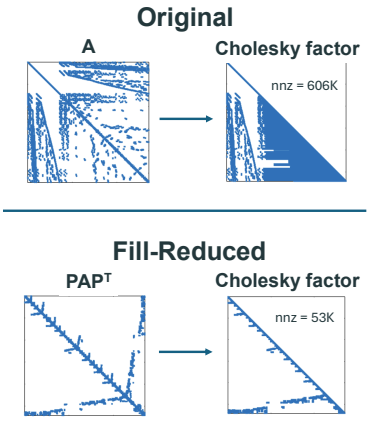


Figure 5: One branch of the design space of parallel symbolic factorization for sparse direct solvers.

Parallelizing the Approximate Minimum Degree Algorithm

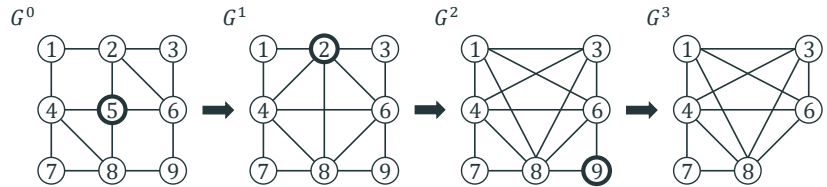
Fill-reduced Ordering is Necessary for Solving Sparse Linear Systems

- **Reducing fill-in** is necessary for solving sparse linear systems.
- Nested dissection [Geo73] and the **approximate minimum degree (AMD) algorithm** [ADD96] are the two heuristics being widely used nowadays.



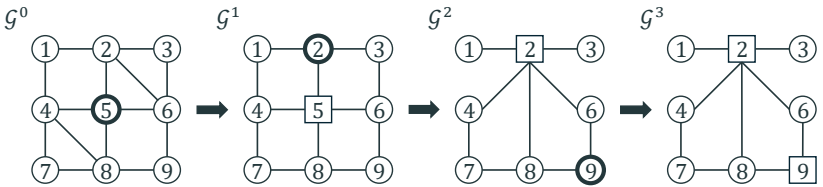
The AMD Algorithm: Elimination Graphs

- Sparse Cholesky factorization can be represented using **elimination graphs**, where a rank-1 update is equivalent to removing a vertex and adding a clique in its neighborhood.
- To reduce fill-in, the heuristic is to **select a minimum degree vertex as the pivot** to eliminate at each step.



The AMD Algorithm: Quotient Graphs

- To represent the sparsity pattern more efficiently, **quotient graphs** represent cliques more compactly—storing only the set of vertices in each clique.
- The neighborhood of the elimination graph can be determined from the original matrix and the clique information stored in the quotient graph.



The AMD Algorithm: Degree Approximation

- To mitigate the **degree update bottleneck**, the AMD algorithm estimates an upper bound for the exact degree using three heuristics:
  - the size of the remaining submatrix
  - the worst-case fill-in
  - union bound without double counting the neighborhood of the pivot

Why Is It Hard to Parallelize the AMD Algorithm?

- A straightforward approach to extract parallelism for the AMD algorithm is to parallelize using **atomic operations**. But there is **limited amount of work with high contention**.

Algorithm 2.1 Computation of $|\mathcal{L}_e \setminus \mathcal{L}_p|$

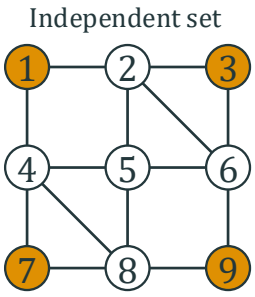
```
1: Assume w(e) < 0 for all e ∈ V̄.
2: for each variable v ∈ L_p do
3:     for each element e ∈ E_v do
4:         if w(e) < 0 then
5:             w(e) ← |L_e|
6:         w(e) ← w(e) − 1
```

Table 3: Average sizes of the sets across all elimination steps.

Matrix Name	$\lvert\mathcal{L}_p\rvert$	$\sum_{v\in\mathcal{L}_p}\lvert\mathcal{E}_v\rvert$	$\left\lvert\bigcup_{v\in\mathcal{L}_p}\mathcal{E}_v\right\rvert$
nd24k	329.7	587.5	14.0
Flan_1565	43.8	64.8	10.2
nlpkkt240	80.5	542.8	56.3

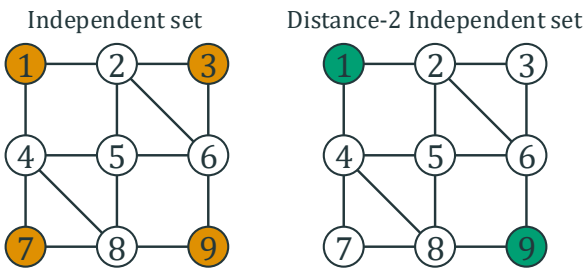
Why Is It Hard to Parallelize the AMD Algorithm?

- The **multiple minimum degree** algorithm [Liu85] eliminates a maximal independent set of pivots with a consolidated degree update afterward.
- In the original design, performance gains from multiple elimination stemmed from the **overlap of pivots' neighborhoods**.
- However, such overlap is not well-suited to the AMD algorithm due to:
  - high contention
  - complex degree approximation



Contribution 1: Multiple Elimination via Distance-2 Independent Sets

- To avoid high contention and complex degree approximation, we exploit **multiple elimination via distance-2 independent sets**.
- This allows each pivot to work independently in both quotient graph and degree updates.



Contribution 2: Relaxation of the Minimum Degree Criteria

- To get more parallelism, we **relax the minimum degree criteria by a multiplicative factor** $mult$.
- This indeed affects the reordering quality, but in a controllable manner as shown in our paper [**C**BD26].

Table 4: Average sizes of maximal distance-2 independent sets across all elimination steps.

Matrix Name	$mult = 1.0$	$mult = 1.1$	$mult = 1.2$
nd24k	2.2	9.0	10.9
Flan_1565	42.0	448.5	678.1
nlpkkt240	57.5	4084.5	6695.8

Other Contributions

To make the full AMD algorithm parallelized, we also implemented the following techniques:

- **Concurrent Quotient Graph Updates:** We redesigned the quotient graph data structure to allow concurrent updates.
- **Duplicated Degree Lists:** To mitigate memory contention during pivot selection, we duplicated the approximate degree lists.
- **Parallel Distance-2 Independent Sets:** We used an analog of the Luby's randomized maximal independent set algorithm [Lub85].

Comprehensive implementation details are available in our paper [**C**BD26].

Evaluation: Environment and Matrix Suite

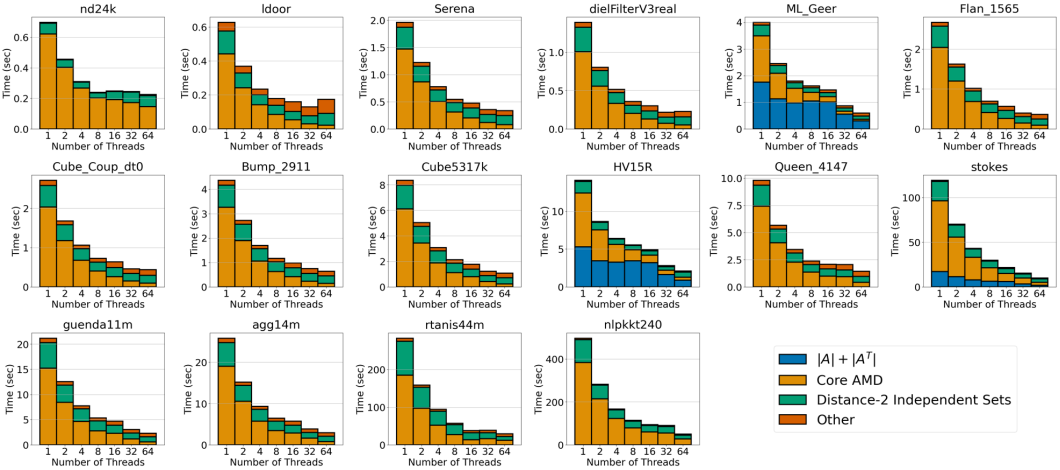
- We evaluated the performance on an AMD EPYC 7763 CPU with 64 cores.

Matrix	#rows	#nonzeros	Symmetric	Positive-definite	Description
nd24k	72.0K	28.7M	✓	✓	3D mesh problem
ldoor	952K	42.5M	✓	✓	Structural problem
Serena	1.39M	64.5M	✓	✓	Structural problem
dielFilterV3real	1.10M	89.3M	✓	×	Electromagnetic problem
ML_Geer	1.50M	111M	×	×	Poroelastic problem
Flan_1565	1.56M	114M	✓	✓	Structural problem
Cube_Coup_dt0	2.16M	124M	✓	×	Structural problem
Bump_2911	2.91M	128M	✓	✓	Geomechanical problem
Cube5317k	5.32M	223M	✓	✓	Elasticity problem
HV15R	2.02M	283M	×	×	Fluid dynamics problem
Queen_4147	4.15M	317M	✓	✓	Structural problem
stokes	11.4M	349M	×	×	Semiconductor process problem
guenda11m	11.5M	512M	✓	✓	Geomechanical problem
agg14m	14.1M	633M	✓	✓	Mesoscale problem
rtanis44m	44.8M	748M	✓	✓	Diffusion problem
nlpkkt240	28.0M	761M	✓	×	Optimization problem

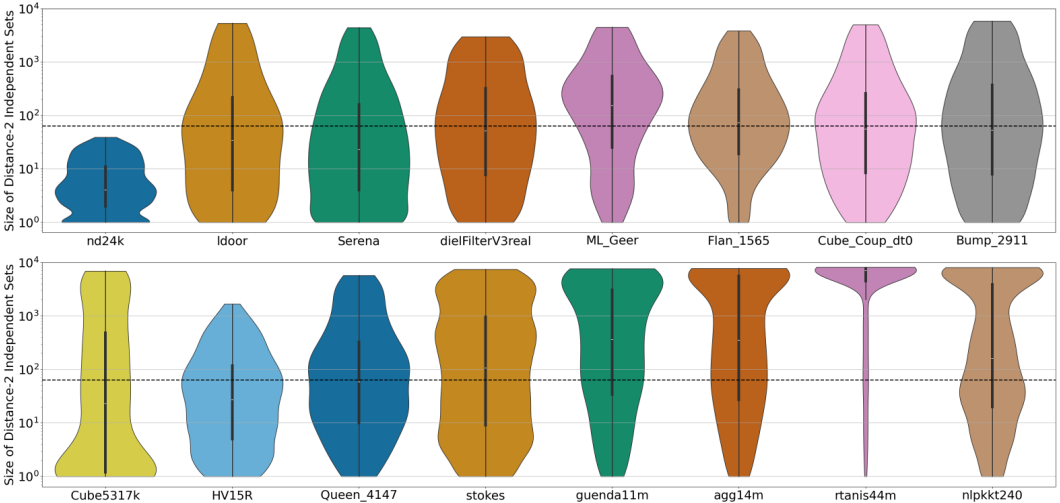
Evaluation: Comparison of Ordering Time and Quality

Matrix	Ordering Time (sec)		Our 64-thread Speedup over SuiteSparse	#Fill-ins		Our Fill-in Ratio
	SuiteSparse	Ours		SuiteSparse	Ours	
nd24k	0.82 ± 0.00	0.26 ± 0.00	(3.18 ± 0.04)×	5.03e+08	5.18e+08	1.02×
ldoor	1.42 ± 0.05	0.32 ± 0.03	(4.39 ± 0.24)×	1.52e+08	1.57e+08	1.04×
Serena	2.81 ± 0.17	0.56 ± 0.05	(5.06 ± 0.20)×	7.48e+09	8.79e+09	1.19×
dielFilterV3real	2.96 ± 0.33	0.54 ± 0.12	(5.59 ± 0.54)×	9.60e+08	1.11e+09	1.16×
ML_Geer	4.26 ± 0.48	0.77 ± 0.20	(5.71 ± 0.77)×	1.42e+09	1.46e+09	1.03×
Flan_1565	4.85 ± 0.56	0.95 ± 0.27	(5.31 ± 0.80)×	3.71e+09	3.94e+09	1.06×
Cube_Coup_dt0	5.21 ± 0.65	1.15 ± 0.30	(4.65 ± 0.58)×	2.24e+10	2.52e+10	1.15×
Bump_2911	6.78 ± 0.74	1.52 ± 0.50	(4.76 ± 0.91)×	4.27e+10	4.83e+10	1.15×
Cube5317k	13.94 ± 0.92	2.90 ± 0.67	(4.99 ± 0.89)×	6.29e+09	6.54e+09	1.04×
HV15R	14.43 ± 1.31	2.34 ± 0.33	(6.23 ± 0.66)×	5.57e+10	5.63e+10	1.01×
Queen_4147	15.45 ± 1.81	2.87 ± 0.79	(5.61 ± 0.82)×	7.70e+10	8.43e+10	1.10×
stokes	85.35 ± 6.51	14.31 ± 4.81	(6.47 ± 1.52)×	7.79e+10	8.17e+10	1.06×
guenda11m	36.59 ± 2.55	8.53 ± 2.47	(4.64 ± 1.27)×	3.10e+11	3.68e+11	1.17×
agg14m	56.10 ± 9.38	10.63 ± 2.99	(5.62 ± 1.38)×	3.75e+11	4.04e+11	1.09×
rtanis44m	195.56 ± 26.25	42.76 ± 12.06	(4.80 ± 0.92)×	1.12e+12	1.25e+12	1.13×
nlpkkt240	393.36 ± 57.62	56.31 ± 15.18	(7.29 ± 1.45)×	1.60e+12	1.82e+12	1.14×

Evaluation: Time Breakdown



Evaluation: Sizes of Distance-2 Independent Sets



End-to-End Comparison for Solving Sparse SPD Systems

Solving SPD systems using cuDSS (v0.7.1) [Nv26] on an A100-80GB GPU in double precision, with different reordering methods:

- **Our parallel AMD** vs. **SuiteSparse AMD** [ADD04]

Matrix	Ours				SuiteSparse AMD			
	Ordering Time (sec)	GPU Solver Time (sec)	Overall Time (sec)	#Fill-ins	Ordering Time (sec)	GPU Solver Time (sec)	Overall Time (sec)	#Fill-ins
`nd24k`	0.26	1.97	2.23	5.18e+08	0.82	1.97	2.79	5.03e+08
`ldoor`	0.32	3.07	3.39	1.57e+08	1.42	3.03	4.45	1.52e+08
`Flan_1565`	0.95	19.62	20.57	3.94e+09	4.85	18.92	23.77	3.71e+09
`Cube5317k`	2.90	44.33	47.23	6.54e+09	13.94	43.90	57.84	6.29e+09

End-to-End Comparison for Solving Sparse SPD Systems

Solving SPD systems using cuDSS (v0.7.1) [Nv26] on an A100-80GB GPU in double precision, with different reordering methods:

- **Our parallel AMD** vs. **cuDSS multi-threaded ND** [Nv26]

Matrix	Ours				cuDSS ND (multi-threaded)			
	Ordering Time (sec)	GPU Solver Time (sec)	Overall Time (sec)	#Fill-ins	Ordering Time (sec)	GPU Solver Time (sec)	Overall Time (sec)	#Fill-ins
`nd24k`	0.26	1.97	2.23	5.18e+08	2.38	1.27	3.65	3.21e+08
`ldoor`	0.32	3.07	3.39	1.57e+08	0.67	2.48	3.15	1.41e+08
`Flan_1565`	0.95	19.62	20.57	3.94e+09	2.41	9.21	11.62	1.46e+09
`Cube5317k`	2.90	44.33	47.23	6.54e+09	7.43	33.45	40.88	4.02e+09

Lessons Learned

- Although the AMD algorithm can be parallelized to some extent and it is possible to hybridize ND with AMD to get an MPI+OpenMP implementation, this still does not provide a solution to GPU reordering methods.
- Parallelizing the reordering phase is not enough since symbolic factorization is also a bottleneck.
- Memory is no longer the sole bottleneck: reducing fill-in may not necessarily enhance overall solver performance.

Future Work

Significant but Unrealistic Goal

Given current research trends, developing **general GPU-resident sparse direct solvers** seems like a high-impact direction, as this approach eliminates the communication bottleneck between CPU and GPU.

But this is too ambitious and unrealistic because:

- there is no universal solution to all solvers.
- this might be a life-long research instead of a PhD thesis.

Meaningful and Realistic Goal

Our roadmap begins with the development of **GPU-resident sparse Cholesky solvers**, specifically by **porting the analysis phase to GPU**.

- SPD matrices exhibit simpler structures than unsymmetric matrices.
- The analysis phase can be completely separated from the numerical phase, and it is easier to plug the new design into existing solvers.

GPU Reordering Phase for Sparse Cholesky Factorization

Porting reordering algorithms from the last century directly to GPU seems hard.

- We need to utilize the massive parallelism on GPU.
- Existing reordering methods target reducing fill-in since memory is limited in the past. Is this still the dominant factor nowadays?
- ND looks for vertex separators and AMD looks for dense neighborhoods. Is there any other structure that helps sparse direct solvers?

GPU Reordering Phase for Sparse Cholesky Factorization

Many reordering algorithms exist (not necessarily for reducing fill-in), it is worth studying how they affect the performance of sparse Cholesky factorization and what structure they reveal.

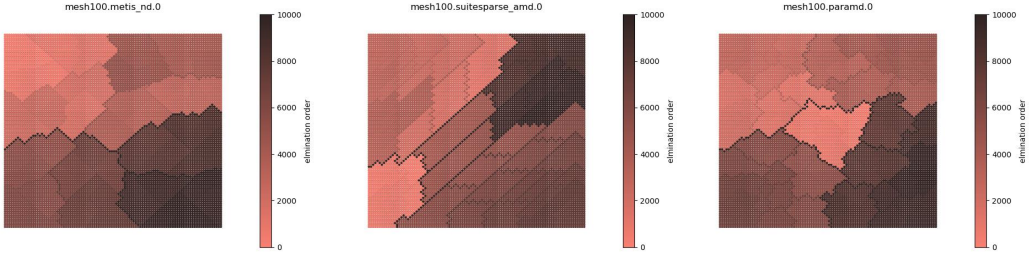


Figure 6: Elimination orders of different reordering methods on a 100-by-100 mesh.

GPU Symbolic Phase for Sparse Cholesky Factorization

A starting point is to extend from previous work [RA23], where the main open problem is that it relies on the elimination tree, which is still computed on CPU.

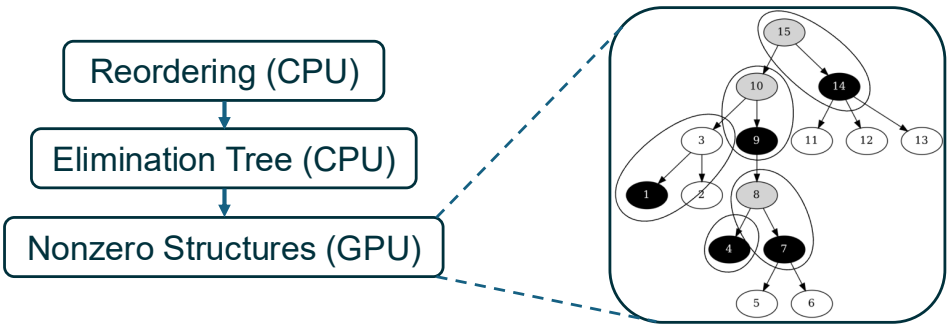


Figure 7: Parallel symbolic factorization scheme used in [RA23].

GPU Symbolic Phase for Sparse Cholesky Factorization

One potential direction is to fuse the elimination tree computation with symbolic factorization, such as the one proposed in [KKB92]. Moreover, we can try to fuse symbolic factorization with the reordering phase, since much information is shared.

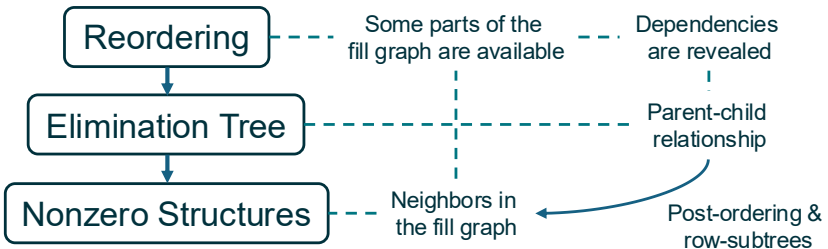


Figure 8: A reason for fusing reordering and symbolic phases.

What If...

What if at the end we find out that the analysis phase is not suitable for GPU?

- In the worst-case scenario, we expect our research to identify the specialized hardware units required on GPUs to efficiently execute the analysis phase, similar to the invention of tensor cores and dynamic programming units.
- On the other hand, we can specialize to applications that exhibit explicit structures, instead of focusing on a generic GPU-resident sparse Cholesky solver.

Conclusions

Summary

- Analysis phase has become a bottleneck for sparse direct solvers.
- We parallelized the AMD algorithm on a single CPU, but there is no hope on porting it to GPU.
- Planned work: GPU-resident sparse Cholesky factorization by designing GPU-friendly reordering methods and GPU-based symbolic factorization.

References i

[CBD26] Yen-Hsiang Chang, Aydın Buluç, and James Demmel. **“Parallelizing the Approximate Minimum Degree Ordering Algorithm: Strategies and Evaluation”.** In: *Proceedings of the 2026 SIAM Conference on Parallel Processing for Scientific Computing (PP)*. SIAM. 2026, pp. 1–15.

[ADD04] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. **“Algorithm 837: AMD, an approximate minimum degree ordering algorithm”.** In: *ACM Transactions on Mathematical Software (TOMS)* 30.3 (2004), pp. 381–388.

[ADD96] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. **“An approximate minimum degree ordering algorithm”.** In: *SIAM Journal on Matrix Analysis and Applications* 17.4 (1996), pp. 886–905.

References ii

[AGB+22] Ahmad Abdelfattah, Pieter Ghysels, Wajih Boukaram, Stanimire Tomov, Xiaoye Sherry Li, and Jack Dongarra. **“Addressing irregular patterns of matrix computations on GPUs and their impact on applications powered by sparse direct solvers”.** In: *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2022, pp. 1–14.

[ASKR18] Alexander Amini, Ava Soleimany, Sertac Karaman, and Daniela Rus. **“Spatial uncertainty sampling for end-to-end control”.** In: *arXiv preprint arXiv:1805.04829* (2018).

[BJN+23] Julian Bellavita, Mathias Jacquelin, Esmond G Ng, Dan Bonachea, Johnny Corbino, and Paul H Hargrove. **“symPACK: a GPU-capable fan-out sparse Cholesky solver”.** In: *Proceedings of the SC’23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. 2023, pp. 1171–1184.

References iii

[CP08] Cédric Chevalier and François Pellegrini. **“PT-Scotch: A tool for efficient parallel graph ordering”.** In: *Parallel computing* 34.6-8 (2008), pp. 318–331.

[CTN+24] Yuwen Chen, Danny Tse, Parth Nobel, Paul Goulart, and Stephen Boyd. **“CuClarabel: GPU acceleration for a conic optimization solver”.** In: *arXiv preprint arXiv:2412.19027* (2024).

[DEFQ21] Ernesto Dufrechou, Pablo Ezzatti, Manuel Freire, and Enrique S Quintana-Ortí. **“Machine learning for optimal selection of sparse triangular system solvers on GPUs”.** In: *Journal of Parallel and Distributed Computing* 158 (2021), pp. 47–55.

[DK23] Arpan Dasgupta and Pawan Kumar. **“Alpha elimination: Using deep reinforcement learning to reduce fill-in during sparse matrix decomposition”.** In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2023, pp. 472–488.

References iv

[DLWL21] Nan Ding, Yang Liu, Samuel Williams, and Xiaoye S Li. **“A message-driven, multi-GPU parallel sparse triangular solver”.** In: *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*. SIAM. 2021, pp. 147–159.

[DP10] Timothy A Davis and Ekanathan Palamadai Natarajan. **“Algorithm 907: KLU, a direct sparse solver for circuit simulation problems”.** In: *ACM Transactions on Mathematical Software (TOMS)* 37.3 (2010), pp. 1–17.

[Duf04] Iain S Duff. **“MA57—a code for the solution of sparse symmetric definite and indefinite systems”.** In: *ACM Transactions on Mathematical Software (TOMS)* 30.2 (2004), pp. 118–144.

[FDE24] Manuel Freire, Ernesto Dufrechou, and Pablo Ezzatti. **“A new level-set analysis and sparse storage format for the SPTRSV in GPUs”.** In: *2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE. 2024, pp. 59–69.

References v

[FZW+23] Xu Fu, Bingbin Zhang, Tengcheng Wang, Wenhao Li, Yuechen Lu, Enxin Yi, Jianqi Zhao, Xiaohan Geng, Fangying Li, Jingwen Zhang, et al. **“Pangulu: A scalable regular two-dimensional block-cyclic sparse direct solver on distributed heterogeneous systems”.** In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2023, pp. 1–14.

[Geo73] Alan George. **“Nested dissection of a regular finite element mesh”.** In: *SIAM journal on numerical analysis* 10.2 (1973), pp. 345–363.

[GHGF26] Atharva Gondhalekar, Kjetil Haugen, Thomas Gibson, and Wu-chun Feng. **“Mapping Sparse Triangular Solves to GPUs via Fine-Grained Domain Decomposition”.** In: *Proceedings of the 2026 SIAM Conference on Parallel Processing for Scientific Computing (PP)*. SIAM. 2026, pp. 88–102.

References vi

[GHS+22] Alice Gatti, Zhixiong Hu, Tess Smidt, Esmond G. Ng, and Pieter Ghysels. **“Graph Partitioning and Sparse Matrix Ordering using Reinforcement Learning and Graph Neural Networks”.** In: *Journal of Machine Learning Research* 23.303 (2022), pp. 1–28. URL: `http://jmlr.org/papers/v23/21-0644.html`.

[GL93] John R Gilbert and Joseph WH Liu. **“Elimination structures for unsymmetric sparse LU factors”.** In: *SIAM Journal on Matrix Analysis and Applications* 14.2 (1993), pp. 334–352.

[GLL21] Anil Gaihre, Xiaoye Sherry Li, and Hang Liu. **“Gsofa: Scalable sparse symbolic lu factorization on gpus”.** In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2021), pp. 1015–1026.

[Glv17] GLVis Team. ***GLVis: OpenGL Finite Element Visualization Tool.*** `glvis.org`. DOI: `10.11578/dc.20171025.1249`.

References vii

[GMBR24] Michael S Gilbert, Kamesh Madduri, Erik G Boman, and Siva Rajamanickam. **“Jet: Multilevel graph partitioning on graphics processing units”.** In: *SIAM Journal on Scientific Computing* 46.5 (2024), B700–B724.

[GP88] John R Gilbert and Tim Peierls. **“Sparse partial pivoting in time proportional to arithmetic operations”.** In: *SIAM journal on scientific and statistical computing* 9.5 (1988), pp. 862–874.

[GS22] Pieter Ghysels and Ryan Synk. **“High performance sparse multifrontal solvers on modern GPUs”.** In: *Parallel Computing* 110 (2022), p. 102897.

[HSLS24] Zhengding Hu, Jingwei Sun, Zhongyang Li, and Guangzhong Sun. **“Ag-sptrsv: An automatic framework to optimize sparse triangular solve on gpus”.** In: *ACM Transactions on Architecture and Code Optimization* 21.4 (2024), pp. 1–25.

References viii

[HSS08] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. **“Exploring network structure, dynamics, and function using NetworkX”.** In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. 2008, pp. 11–15.

[KK98] George Karypis and Vipin Kumar. **“A parallel algorithm for multilevel graph partitioning and sparse matrix ordering”.** In: *Journal of parallel and distributed computing* 48.1 (1998), pp. 71–95.

[KKB92] P Sreenivasa Kumar, M Kishore Kumar, and A Basu. **“A parallel algorithm for elimination tree computation and symbolic factorization”.** In: *Parallel computing* 18.8 (1992), pp. 849–856.

[Kum24] Dhiraj Kumar. ***What Is Bundle Adjustment?*** `https://www.baeldung.com/cs/computer-vision-bundle-adjustment`.

References ix

[LB12] Oren E Livne and Achi Brandt. **“Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver”.** In: *SIAM Journal on Scientific Computing* 34.4 (2012), B499–B522.

[LGCL21] Yang Liu, Pieter Ghysels, Lisa Claus, and Xiaoye Sherry Li. **“Sparse approximate multifrontal factorization with butterfly compression for high-frequency wave equations”.** In: *SIAM Journal on Scientific Computing* 43.5 (2021), S367–S391.

[Liu85] Joseph WH Liu. **“Modification of the minimum-degree algorithm by multiple elimination”.** In: *ACM Transactions on Mathematical Software (TOMS)* 11.2 (1985), pp. 141–153.

[LJGL18] Yang Liu, Mathias Jacquelin, Pieter Ghysels, and Xiaoye S Li. **“Highly scalable distributed-memory sparse triangular solution algorithms”.** In: *2018 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing*. SIAM. 2018, pp. 87–96.

References x

[LK15] Dominique LaSalle and George Karypis. **“Efficient nested dissection for multicore architectures”.** In: *European Conference on Parallel Processing*. Springer. 2015, pp. 467–478.

[LLJ+26] Wan Luan Lee, Dian-Lun Lin, Shui Jiang, Cheng-Hsiang Chiu, Yibo Lin, Bei Yu, Tsung-Yi Ho, and Tsung-Wei Huang. **“G-kway: Multilevel gpu-accelerated k-way graph partitioner using task graph parallelism”.** In: *ACM Transactions on Design Automation of Electronic Systems* 31.3 (2026), pp. 1–26.

[LLLS23] Xiaoye S Li, Paul Lin, Yang Liu, and Piyush Sao. **“Newly released capabilities in the distributed-memory superlu sparse direct solver”.** In: *ACM Transactions on Mathematical Software* 49.1 (2023), pp. 1–20.

References xi

[LNL20] Zhengyang Lu, Yuyao Niu, and Weifeng Liu. **“Efficient block algorithms for parallel sparse triangular solve”.** In: *Proceedings of the 49th International Conference on Parallel Processing*. 2020, pp. 1–11.

[LNY+26] Ziwei Li, Shuzi Niu, Tao Yuan, Huiyuan Li, and Wenjia Wu. **“Factorization-in-Loop: Proximal Fill-in Minimization for Sparse Matrix Reordering”.** In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 40. 1. 2026, pp. 730–737.

[Lub85] Michael Luby. **“A simple parallel algorithm for the maximal independent set problem”.** In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1985, pp. 1–10.

References xii

[LVW00] Sebastien Lacroix, Yuri V Vassilevski, and Mary F Wheeler. **“Iterative solvers of the implicit parallel accurate reservoir simulator (IPARS), I: single processor case”.** In: *TICAM report 00-28, The University of Texas at Austin* (2000).

[LXG+21] Yang Liu, Xin Xing, Han Guo, Eric Michielssen, Pieter Ghysels, and Xiaoye Sherry Li. **“Butterfly factorization via randomized matrix-vector multiplications”.** In: *SIAM Journal on Scientific Computing* 43.2 (2021), A883–A907.

[LZ20] Ruipeng Li and Chaoyu Zhang. **“Efficient parallel implementations of sparse triangular solves for GPU architectures”.** In: *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing*. SIAM. 2020, pp. 106–117.

References xiii

[LZN+26] Yida Li, Siwei Zhang, Yiduo Niu, Yang Du, Qingxiao Sun, Zhou Jin, and Weifeng Liu. **“Trojan Horse: Aggregate-and-Batch for Scaling Up Sparse Direct Solvers on GPU Clusters”.** In: *Proceedings of the 31st ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 2026, pp. 369–383.

[NAA25] Pratik Nayak, Isha Aggarwal, and Hartwig Anzt. **“Efficient solution of batched band linear systems on GPUs”.** In: *The International Journal of High Performance Computing Applications* 39.5 (2025), pp. 615–630.

[Nv26] NVIDIA Corporation. ***NVIDIA cuDSS (Preview): A high-performance CUDA Library for Direct Sparse Solvers.*** `https://docs.nvidia.com/cuda/cudss/index.html`.

References xiv

[PDF+18] Grégoire Pichon, Eric Darve, Mathieu Faverge, Pierre Ramet, and Jean Roman. **"Sparse supernodal solver using block low-rank compression: Design, performance and analysis".** In: *Journal of computational science* 27 (2018), pp. 255–270.

[PFRR17] Grégoire Pichon, Mathieu Faverge, Pierre Ramet, and Jean Roman. **"Reordering strategy for blocking optimization in sparse linear solvers".** In: *SIAM Journal on Matrix Analysis and Applications* 38.1 (2017), pp. 226–248.

[PG25] Pierre-Etienne Polet and Thierry Gautier. **"On the Use of APU Architectures in MUMPS/XKBlas".** In: *MUMPS workshop*. 2025, pp. 1–37.

[PT20] Shaoyi Peng and Sheldon X-D Tan. **"GLU3.0: Fast GPU-based parallel sparse LU factorization for circuit simulation".** In: *IEEE Design & Test* 37.3 (2020), pp. 78–90.

References xv

[RA23] Tobias Ribizel and Hartwig Anzt. **“Parallel symbolic cholesky factorization”.** In: *Proceedings of the SC’23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. 2023, pp. 1721–1727.

[RT78] Donald J Rose and Robert Endre Tarjan. **“Algorithmic aspects of vertex elimination on directed graphs”.** In: *SIAM Journal on Applied Mathematics* 34.1 (1978), pp. 176–197.

[RZH+25] Jie Ren, Tingxuan Zhong, Yuxi Hong, Guofeng Feng, Xincheng Wang, Weile Jia, Hatem Ltaief, and David Elliot Keyes. **“Caracal: A GPU-Resident Sparse LU Solver with Lightweight Fine-Grained Scheduling”.** In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2025, pp. 1477–1494.

[Sch82] Robert Schreiber. **“A new implementation of sparse Gaussian elimination”.** In: *ACM Transactions on Mathematical Software (TOMS)* 8.3 (1982), pp. 256–276.

References xvi

[SKLV19] Piyush Sao, Ramakrishnan Kannan, Xiaoye Sherry Li, and Richard Vuduc. **"A communication-avoiding 3D sparse triangular solver".** In: *Proceedings of the ACM International Conference on Supercomputing*. 2019, pp. 127–137.

[ŚKR+24] Kasia Świrydowicz, Nicholson Koukpaizan, Tobias Ribizel, Fritz Göbel, Shrirang Abhyankar, Hartwig Anzt, and Slaven Peleš. **"GPU-resident sparse direct linear solvers for alternating current optimal power flow analysis".** In: *International Journal of Electrical Power & Energy Systems* 155 (2024), p. 109517.

[SLB25] Oguz Selvitopi, Xiaoye S Li, and Aydin Buluc. **"Performance-Portable Symbolic Factorization through Common Graph Operations".** In: *Proceedings of the SC'25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2025, pp. 804–812.

References xvii

[SLV18] Piyush Sao, Xiaoye Sherry Li, and Richard Vuduc. **“A communication-avoiding 3D LU factorization algorithm for sparse matrices”.** In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2018, pp. 908–919.

[SLV19] Piyush Sao, Xiaoye S Li, and Richard Vuduc. **“A communication-avoiding 3D algorithm for sparse LU factorization on heterogeneous systems”.** In: *Journal of Parallel and Distributed Computing* 131 (2019), pp. 218–234.

[SLVL15] Piyush Sao, Xing Liu, Richard Vuduc, and Xiaoye Li. **“A sparse direct solver for distributed memory Xeon Phi-accelerated systems”.** In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 71–81.

References xviii

[TJC+25] Tao Tang, Youfu Jiang, Yingbo Cui, Jianbin Fang, Peng Zhang, Lin Peng, and Chun Huang. **"Selection of Supervised Learning-based Sparse Matrix Reordering Algorithms".** In: *arXiv preprint arXiv:2511.10180* (2025).

[XJAR23] Yang Xia, Peng Jiang, Gagan Agrawal, and Rajiv Ramnath. **"End-to-End LU factorization of large matrices on GPUs".** In: *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 2023, pp. 288–300.

[ZMD+26] Behrooz Zarebavami, Ahmed H Mahmoud, Ana Dodik, Changcheng Yuan, Serban D Porumbescu, John D Owens, Maryam Mehri Dehnavi, and Justin Solomon. **"Fast Sparse Matrix Permutation for Mesh-Based Direct Solvers".** In: *arXiv preprint arXiv:2602.00898* (2026).

References xix

[ZMIM25] Behrooz Zarebavani, Danny M. Kaufman, David IW Levin, and Maryam Mehri Dehnavi. **“Adaptive Algebraic Reuse of Reordering in Cholesky Factorizations with Dynamic Sparsity Patterns”.** In: *ACM Transactions on Graphics (TOG)* 44.4 (2025), pp. 1–17.

Backup Slides

Notation in ParAMD [CBD26]

Quotient graph invariant:

$$N_v = (\mathcal{A}_v) \cup \bigcup_{e \in \mathcal{E}_v} \mathcal{L}_e \setminus \{v\}.$$

Degree approximation:

$$d_v^k = \min \left\{ \begin{array}{l} n - k - 1, \\ d_v^{k-1} + |\mathcal{L}_p \setminus \{v\}| - 1, \\ |\mathcal{A}_v \setminus \{v\}| + |\mathcal{L}_p \setminus \{v\}| + \sum_{e \in \mathcal{E}_v} |\mathcal{L}_e \setminus \mathcal{L}_p| \end{array} \right\}$$

Symbol	Description
A	An $n \times n$ sparse symmetric matrix
p	The selected pivot
G^k	The kth elimination graph
$\mathcal{G}^k$	The kth quotient graph
V^k	The set of vertices in G^k, equivalent to the set of variables in $\mathcal{G}^k$
E^k	The set of edges in G^k
$\overline{V}^k$	The set of elements in $\mathcal{G}^k$
N_v^k	The neighborhood of vertex v in G^k
$\mathcal{A}^k$	The connections among V^k in $\mathcal{G}^k$
$\mathcal{E}^k$	The connections from V^k to $\overline{V}^k$ in $\mathcal{G}^k$
$\mathcal{L}^k$	The connections from $\overline{V}^k$ to V^k in $\mathcal{G}^k$
$\mathcal{A}_v^k$	The variables adjacent to variable v in $\mathcal{G}^k$
$\mathcal{E}_v^k$	The elements adjacent to variable v in $\mathcal{G}^k$
$\mathcal{L}_e^k$	The variables adjacent to element e in $\mathcal{G}^k$

Figure 9: Notation table.